ATMIC NETWORKS
YOUR GUIDE TO GLOBAL CERTIFIATION

"Please note that these files may not be up to date. However, the questions will help you understand the exam format and typical question patterns."

www.atmicnetworks .com

Warning: Keep connected with our support team for latest updates

Question: 1

Although Salesforce B2B Commerce and Salesforce recommend against using "without sharing classes" whenever possible, sometimes it is unavoidable. Which three items will open up a major security hole? (3 answers)

A. Executing dynamic SOQL inside a without sharing class with a bind variable fromPageReference.getParameters().
B. Executing dynamic SOQL inside a without sharing class with a bind variable from theUserInfo class.
C. Executing dynamic SOQL inside a without sharing class with a bind variable fromPageReference.getCookies().
D. Executing dynamic SOQL inside a without sharing class with a bind variable fromcc_RemoteActionContentex class.
E. Executing dynamic SOQL inside a without sharing class with a bind variable fromccAPI.CURRENT_VERSION.

Answer: A,C,D

Explanation:

Executing dynamic SOQL inside a without sharing class with a bind variable from PageReference.getParameters(), PageReference.getCookies(), or cc_RemoteActionContext class will open up a major security hole because these sources of input are not sanitized and can be manipulated by malicious users to inject SOQL queries that bypass the sharing rules and access data that they are not supposed to see. For example, a user can modify the URL parameters or cookies to include a SOQL query that returns sensitive data from the database. To prevent this, it is recommended to use static SOQL or escape the bind variables before executing dynamic SOQL.

Question: 2

The ccrz.cc_hk_UserInterface apex class, HTMLHead Include Begin and HTML Head Include End Cloudcraze Page Include sections allow additional content to be added to the HTML <head> tag. What are two reasons that is it preferred to use the ccrz.cc_hk_UserInterface extension over the Cloudcraze Page Include sections? (2 answers)

A. Salesforce apex:include is wrapped in tags.
B. HTML does not support tags inside the <head>
C. Salesforce apex:include is wrapped in tags.

D. HTML does not support tags inside the <head>

Answer: A,D

Explanation:

The ccrz.cc_hk_UserInterface apex class is preferred over the HTMLHead Include Begin and HTML Head Include End Cloudcraze Page Include sections because Salesforce apex:include is wrapped in tags, which are not valid inside the HTML <head> tag. This can cause rendering issues or unexpected behavior in some browsers. The ccrz.cc_hk_UserInterface extension allows adding content to the HTML <head> tag without using apex:include.

Question: 3

The ccUtil apex class in Salesforce B2B Commerce provides numerous utility functions that can be leveraged in subscriber classes.
What are two ways to check the input or return data of the Global API's? (2 answers)

A. ccrz.ccUtil.isNotEmpty(Map<String, Object>) andccrz.ccUtil.isNotEmpty(List<Object>)
B. ccrz.ccUtil.isNotValid(Map<String, Object>) andccrz.ccUtil.isNotValid(List<Object>)
C. ccrz.ccUtil.isValid(Map<String, Object>) and ccrz.ccUtil.isValid(List<Object>)
D. ccrz.ccUtil.isEmpty(Map<String, Object>) and ccrz.ccUtil.isEmpty(List<Object>)

Answer: A,D

Explanation:

The ccUtil apex class provides two methods to check the input or return data of the Global API's: ccrz.ccUtil.isNotEmpty(Map<String, Object>) and ccrz.ccUtil.isEmpty(Map<String, Object>). These methods return true if the map is not null and contains at least one entry, or if the map is null or empty, respectively. Similarly, ccrz.ccUtil.isNotEmpty(List<Object>) and ccrz.ccUtil.isEmpty(List<Object>) return true if the list is not null and contains at least one element, or if the list is null or empty, respectively. These methods are useful for validating the input parameters or the output results of the Global API's.

Question: 4

The ccUtil apex class in Salesforce B2B Commerce provides numerous utility functions that can be leveraged in subscriber classes. Which command will return the value in the given Map if found or a default value in the event that the Map is null, empty, or an object is not found for that key?

A. ccrz.ccUtil.defv (Map<String.Object> mp, String key , Object ob)
B. ccrz.ccUtil.defVal (Map<String.Object> mp, String key, Object ob)
C. ccrz.ccUtil .. (Map<String.Object> mp, String key, Object ob)
D. ccrz.ccUtil.defaultValue(Map<String.Object> mp, String key , Object ob)

Answer: B

Explanation:

The ccrz.ccUtil.defVal (Map<String.Object> mp, String key, Object ob) method will return the value in the given Map if found or a default value in the event that the Map is null, empty, or an object is not found for that key. This method is useful for providing fallback values for configuration settings or input parameters that may be missing or invalid. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, ccUtil Class

Question: 5

A configuration value, CO.NewOrder, is set to TRUE. What is one way of preventing anexisting payment page from being shown on the checkout payment page?

A. Delete the Visualforce page from the code base.
B. Remove the value matching the page name from the pmt.whitelist configurationsetting, then rebuild and activate

a new Configuration cache

C. Remove the payment type associated with the payment page from CO.pmts, thenrebuild and activate a new cache.

D. Override the front end template and modify the way the embedded payment page getsloaded from the payment list configuration.

Answer: B

Explanation:

This approach effectively removes the payment page from the list of allowed pages, ensuring it is not displayed during the checkout process.
Reference: Salesforce B2B Commerce documentation on checkout process customization and configuration settings, specifically focusing on payment page handling and the pmt.whitelist setting.

The pmt.whitelist configuration setting in Salesforce B2B Commerce is used to manage the Visualforce pages that support all payment types on the storefront1. If you want to prevent an existing payment page from being shown on the checkout payment page, one way to do it is to remove the value matching the page name from the pmt.whitelist configuration setting. After doing this, you would need to rebuild and activate a new Configuration cache for the changes to take effect1. Please note that this information is based on the Salesforce B2B Commerce documentation and best practices1.

Question: 6

A Developer created a custom field that a project wants to expose on a given page. How does the Developer ensure that the field is available to display on a given page?

A. Override the Service Class that the page uses and update the ServiceManagementin CCAdmin for the given storefront to use this new Service Class.

B. Override the Logic Class that the page uses and update the Service Management inCCAdmin for the given storefront to use this new Service Class

C. Create a new Service Classthat the page uses and update the Service Managementin CCAdmin for the given storefront to use this new Service Class

D. Create a new Logic Class that the page uses and update the Service Managementin CCAdmin for the given storefront to use this new Service Class

Answer: A

Explanation:

To ensure that a custom field is available to display on a given page, the Developer needs to override the Service Class that the page uses and update the Service Management in CCAdmin for the given storefront to use this new Service Class. The Service Class is responsible for retrieving and setting data for a given page. The Logic Class is responsible for implementing business logic and validation rules for a given page. Creating a new Service Class or Logic Class is not necessary, as overriding an existing one can achieve the same result. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Service Classes, Logic Classes

Question: 7

A developer is trying to troubleshoot why a field is not displaying on the Product Detail Page. What should be typed in the Developer Tools Console in the browser to view the fields available for the Product Detail Page?

A. CCRZ.productSearchView
B. CCRZ.cartView
C. CCRZ.productDetailModel
D. CCRZ.productDetailView

Answer: C

Explanation:

To view the fields available for the Product Detail Page, the developer should type CCRZ.productDetailModel in the Developer Tools Console in the browser. This will display the product detail model object, which contains the product data and attributes that are rendered on the page. The other options are either not valid or not relevant for the Product Detail Page.

Question: 8

For which two reasons is it preferable to extend the Salesforce B2B Commerce remote invocation object instead of using the standard Salesforce remote action invocation manager (2 answers)

A. A standard remote action will not have access to Salesforce B2B Commerce objects.
B. The APEX method called by the remote action will be passed as a Salesforce B2B Commerce context object.
C. Salesforce B2B Commerce includes do not support standard SalesForce remote actions.
D. The Salesforce B2B Commerce logger cannot be utilized in standard remote actions

Answer: B,D

Explanation:

It is preferable to extend the Salesforce B2B Commerce remote invocation object instead of using the standard Salesforce remote action invocation manager for two reasons:
The APEX method called by the remote action will be passed as a Salesforce B2B Commerce context object, which contains useful information such as the current user, cart, storefront, and configuration settings. This can simplify the development and testing of the remote action.
The Salesforce B2B Commerce logger can be utilized in the remote action, which allows logging messages and errors to the debug log or to a custom object. This can facilitate debugging and troubleshooting of the remote action.

Question: 9

How are variables bound when services use the ccSercviceDao classto execute queries?

A. Global variables
B. Apex local variables
C. String substitution
D. Apex class variables

Answer: C

Explanation:

When services use the ccServiceDao class to execute queries, variables are bound by string substitution. This means that the query string contains placeholders for variables that are replaced by their values at runtime. For example, ccrz.ccServiceDao.getQuery('SELECT Id FROM Account WHERE Name = :name') will replace :name with the value of the name variable.

Question: 10

How are version related upgrades passed on to subscriber API extensions/overrides?

A. APIs callback with specific versions specified; the user must know which version number to use.
B. Copy and paste of specific code is "built-in"
C. Extensions and overridden APIs don't support-related upgrades.
D. The "delegate" allows inherited method calls access to the most recentlyspecified service version

Answer: D

Explanation:

Version related upgrades are passed on to subscriber API extensions/overrides by using the “delegate” keyword, which allows inherited method calls access to the most recently specified service version. For example, delegate.getCart() will invoke the getCart() method of the latest service version that is available for the current storefront. This way, extensions and overrides can leverage the new features and enhancements of the upgraded service versions without modifying their code.

Question: 11

How can the display of CC Menu Items be customized for different users?

A. cc_hk_Category extension to pre-process which category items are cached as menu items
B. cc_hk_Menu extension to post-process any cached menu items
C. cc_hk_Menu extension to pre-process which menu items are cached
D. cc_hk_Category extension to post-process any cached menu items

Answer: B

Explanation:

The display of CC Menu Items can be customized for different users by using the cc_hk_Menu extension to post-process any cached menu items. This extension allows modifying the menu items based on the user context, such as the user role, account, or cart. For example, the extension can hide or show certain menu items based on the user’s permissions or preferences.

Question: 12

How does a project implement the process to persist payment information datain the Checkout flow for Salesforce B2B Commerce version 4.2 and beyond?

A. Trigger a remote action when the process payment button is selected to capture the payment.
B. Trigger a remote action to store the payment information in the URL query parameters.
C. Trigger the processPayment event and pass in the payment information object as an argument.
D. Trigger the externalprocessedPayment and pass in the payment information object as an argument.

Answer: C

Explanation:

To persist payment information data in the Checkout flow for Salesforce B2B Commerce version 4.2 and beyond, the project needs to trigger the processPayment event and pass in the payment information object as an argument. This event will invoke the processPayment method of the ccServicePayment class, which will validate and process the payment information and return a payment result object. The payment result object will contain the status and details of the payment transaction.

Question: 13

How do the REST APIs in Salesforce B2B Commerce support pass-through parameter handling

A. An exception is generated for unknown API keys
B. Parameters are passed through the service handlers
C. Parameters are filtered out before the request is processed
D. Parameters are separated, but unused

Answer: B

Explanation:

The REST APIs in Salesforce B2B Commerce support pass-through parameter handling by passing parameters through the service handlers. This means that any parameters that are not recognized by the REST API framework will be passed to the service handler class that implements the API logic.
The service handler class can then use these parameters for custom logic or validation.

Question: 14

How is a price group dynamically set?

A. By overriding the ccLogicProductPrice class
B. By using contract pricing
C. By extending the ccApiPriceList API
D. By extending the cc_hk_priceing hook

Answer: D

Explanation:

A price group can be dynamically set by extending the cc_hk_pricing hook. This hook allows modifying the price group based on various factors, such as the user, cart, product, or storefront. For example, the hook can apply a different price group for a specific product category or for a specific user segment.

Question: 15

Inwhich three different ways can a theme be enabled in Salesforce B2B Commerce? (3 answers)

A. A Storefront setting
B. An Account Group field value
C. A per user setting
D. Account
E. Dynamically through a hook

Answer: A,B,E

Explanation:

A theme can be enabled in Salesforce B2B Commerce in three different ways:
A Storefront setting: The theme can be specified in the Storefront Configuration settings in CCAdmin. This will apply the theme to all users who access the storefront.
An Account Group field value: The theme can be specified in the Theme field of an Account Group record in Salesforce. This will apply the theme to all users who belong to that account group. Dynamically through a hook: The theme can be specified dynamically by extending the cc_hk_theme hook. This hook allows changing the theme based on various factors, such as the user, cart, product, or storefront. For example, the hook can apply a different theme for a specific product category or for a specific user segment.

Question: 16

In which threeways can Salesforce B2B Commerce API sizing blocks support multiple API sizing requests? (3 answers)

A. When different entities are specified in the method invocation.
B. The sizing block is not removed.
C. SZ_ASSC is used.
D. The sizing block is removedafter the first handler.
E. SZ_ASSC is not used.

Answer: A, C, E

Explanation:

Salesforce B2B Commerce API sizing blocks can support multiple API sizing requests in three ways: When different entities are specified in the method invocation. For example, ccrz.ccServiceProduct.getProducts(ccrz.ccAPI.SZ_M, ccrz.ccAPI.SZ_L) will use the SZ_M sizing block for the product entity and the SZ_L sizing block for the related entities.
SZ_ASSC is used. This flag indicates that the associated entities should use the same sizing block as the primary entity. For example, ccrz.ccServiceProduct.getProducts(ccrz.ccAPI.SZ_M, ccrz.ccAPI.SZ_ASSC) will use the SZ_M sizing block for both the product entity and the related entities.
SZ_ASSC is not used. This flag indicates that the associated entities should use the default sizing block, which is SZ_M, unless otherwise specified. For example, ccrz.ccServiceProduct.getProducts(ccrz.ccAPI.SZ_M) will use the SZ_M sizing block for the product entity and the SZ_M sizing block for the related entities. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Data Sizing Conventions

Question: 17

In which three ways should useful debugging information in Salesforce B2B
Commerce implementation be garnered? (3 answers) A) Enabling the logging token via

A. Admin andsubsequently inspecting the logs via the browser console.
B. Logging a case with Salesforce support to enable advanced debugging options.
C. Enabling debugging options for the current user and visually inspecting the Salesforce debug logs.
D. Placing a System.debug() statement anywhere in the class being debugged.
E. Logging into the community as a system administrator to identify any potential permissions or Visualforce exceptions.

Answer: A,C, E

Explanation:

Useful debugging information in Salesforce B2B Commerce implementation can be garnered in three ways:
Enabling the logging token via Admin and subsequently inspecting the logs via the browser console.
This will enable logging messages and errors to the browser console, which can be viewed by opening the Developer Tools in the browser. The logging token can be enabled by setting the value of
CO.logToken to true in CCAdmin.
Enabling debugging options for the current user and visually inspecting the Salesforce debug logs. This will enable logging messages and errors to the Salesforce debug logs, which can be viewed by opening the Debug Logs page in Salesforce Setup. The debugging options can be enabled by creating a Debug Level and a Trace Flag for the current user in Salesforce Setup.
Logging into the community as a system administrator to identify any potential permissions or Visualforce exceptions. This will allow viewing any errors or warnings that may occur on the community pages due to insufficient permissions or Visualforce issues. The system administrator can also access CCAdmin and other tools from within the community.
Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Logging, Debug Your Code

Question: 18

A new payment type for the Checkout flow has been implemented. Which three descriptors follow best practice for possible configuration metadata are needed to enable a flow? (3 answers)

A. *.pay
B. Cart
C. Checkout
D. *.Edit
E. *.New

Answer: A,D,E

Explanation:

To enable a new payment type for the Checkout flow, three possible configuration metadata descriptors are needed:
*.pay: This descriptor defines the payment type name, label, description, icon, and handler class. For example, CO.pmts.CreditCard.pay defines the payment type for credit card payments.
*.Edit: This descriptor defines the Visualforce page that is used to edit or update an existing payment of this type. For

example, CO.pmts.CreditCard.Edit defines the page that allows editing a credit card payment.
*.New: This descriptor defines the Visualforce page that is used to create a new payment of this type. For example, CO.pmts.CreditCard.New defines the page that allows creating a new credit card payment. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Payment Configuration Settings

Question: 19

Numerous flags ... have a directimpact on the result set provided by the Global
API's. What Global API Data-Sizing convention flag prevents an API request from propagating to further requests when provided as a Boolean parameter with a value of true?

A. ccrz.ccAPI.SZ_REL
B. ccrz.ccAPI.SZ_ASSC
C. ccrz.ccAPISizing.ASSC
D. ccrz.ccAPISizing.REL

Answer: B

Explanation:

The Global API Data-Sizing convention flag that prevents an API request from propagating to further requests when provided as a Boolean parameter with a value of true is ccrz.ccAPI.SZ_ASSC. This flag indicates that only one API request should be executed with the specified sizing block, and any subsequent requests should use their own default sizing blocks. For example, ccrz.ccServiceCart.getCart(ccrz.ccAPI.SZ_L,true) will use the SZ_L sizing block for retrieving the cart data, but any other requests that are triggered by this method will use their own default sizing blocks. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Data Sizing Conventions

Question: 20

Numerous flags, when set, have a direct impact on the result set provided by the Global API's. What is the default Global API DataSizing convention flag that is used by theAPI's unless otherwise specified?

A. CCRZ.ccPAI.SZ_XL
B. CCRZ.ccPAI.SZ_M
C. CCRZ.ccPAI.SZ_L
D. CCRZ.ccPAI.SZ_S

Answer: B

Explanation:

The default Global API Data-Sizing convention flag that is used by the API's unless otherwise specified is CCRZ.ccAPI.SZ_M. This flag indicates that the medium sizing block should be used for retrieving data, which includes the most commonly used fields and related entities. For example, ccrz.ccServiceProduct.getProducts() will use the SZ_M sizing block by default, unless another sizing block is specified as a parameter. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Data Sizing Conventions

Question: 21

Numerous flags when set, have a direct impact on the result set provided by the Global API's. Which conversion flag allows for sObjects to be returned from the Global API's when provided as a Boolean parameter with a value of true?

A. ccrz.ccAPISizing.SKIPTRZ
B. ccrz.ccAPISizing.SOBJECT
C. ccrz.ccAPI.SZ_SKIPTRZ
D. ccrz.ccAPI.SZ_SOBJECT

Answer: D

Explanation:

The conversion flag that allows for sObjects to be returned from the Global API's when provided as a Boolean parameter with a value of true is ccrz.ccAPI.SZ_SOBJECT. This flag indicates that the API should return the raw sObjects instead of the transformed objects that are usually returned by the API. For example, ccrz.ccServiceProduct.getProducts(ccrz.ccAPI.SZ_SOBJECT,true) will return the Product2 sObjects instead of the ccrz E_Product c objects. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Data Sizing Conventions

Question: 22

A query containing a subquery is executed. What is appended to the subquery name as part of its transformation by default in Salesforce B2B Commerce?

A. A subscriber-supplied token
B. " ccrz"
C. The "*" symbol
D. The letter "S"

Answer: B

Explanation:

A query containing a subquery is executed. By default, in Salesforce B2B Commerce, " ccrz" is appended to the subquery name as part of its transformation. This is done to avoid conflicts with the standard Salesforce fields and relationships. For example, SELECT Id, Name, (SELECT Id, Name FROM Contacts) FROM Account will be transformed to SELECT Id, Name, (SELECT Id, Name FROM Contacts ccrz) FROM Account ccrz. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Query Transformation

Question: 23

Salesforce B2B Commerce natively provides a robots.txt file, however, a customer can also create its own version.Which three scenarios are valid reasons for customer to create their own robots.txt file? (3 answers)

A. The customer wants to reference multiple storefront sitemap indexes in a single robots.txt file B. The customer wants to reference a custom sitemap index.

C. The customer wants to have multiple robot.txt files in a single Salesforce Community.
D. The customer's store is not located at the rootof their domain.
E. robot.txt only works if there is one storefront in the org

Answer: A,B,D

Explanation:

A customer can create its own robots.txt file for three valid reasons:
The customer wants to reference multiple storefront sitemap indexes in a single robots.txt file. This

can be useful if the customer has multiple storefronts under the same domain and wants to provide a single entry point for search engines to crawl their sitemaps.
The customer wants to reference a custom sitemap index. This can be useful if the customer has created their own sitemap index that contains custom sitemaps or sitemaps from other sources. The customer's store is not located at the root of their domain. This can be useful if the customer has their store under a subdirectory or a subdomain and wants to specify a different robots.txt file for their store than for their main domain. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Robots.txt File

Question: 24

Salesforce B2B leverages global API's for encapsulating business logic into blocks that can be extended and modified by subscribers. Which three statements are true regarding extending ccServiceProduct and exposing custom fields on the Product Detail Page? (3 answers)

A. Override the getFieldsMap method and add subscriber specific code.
B. Ensure that any helper methods are defined as private andstatic only.
C. Create a global with sharing class that extends ccrz.ccServiceProduct.
D. Create a public with sharing class that extends ccrz.ccServiceProduct.
E. Override the fetch method and add your subscriber specific code here.

Answer: A,C,E

Explanation:

To extend ccServiceProduct and expose custom fields on the Product Detail Page, three statements are true:
Override the getFieldsMap method and add subscriber specific code. This method returns a map of field names and field types for the product entity and its related entities. By overriding this method, the subscriber can add their custom fields to the map and make them available for the API.
Create a global with sharing class that extends ccrz.ccServiceProduct. This class will inherit all the methods and properties of the ccServiceProduct class and allow overriding or extending them. The class should be global and with sharing to ensure that it can be accessed by the API framework and respect the sharing rules.
Override the fetch method and add your subscriber specific code here. This method executes the query to fetch the product data and returns a result object. By overriding this method, the subscriber can modify the query or the result object to include their custom fields or logic. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Service Classes, ccServiceProduct Class

Question: 25

The sizing keys used in the Salesforce B2B Commerce Global APIs five distinct operations. What are three of these

operations? (3 answers)

A. Refetch data (used on some Logic classes)
B. Return formats as Map<String, Object> or SObjects lists
C. Override static DAO classes and methods
D. Related Query to call (sub queries or direct queries)
E. Object type casting

Answer: ADE

Explanation:

The sizing keys used in the Salesforce B2B Commerce Global APIs perform five distinct operations. Three of these operations are:
Refetch data (used on some Logic classes): This operation indicates that the data should be refetched from the database instead of using the cached data. For example, ccrz.ccServiceCart.getCart(ccrz.ccAPI.SZ_REFETCH) will refetch the cart data and refresh the cache.
Related Query to call (sub queries or direct queries): This operation indicates that the related entities should be retrieved by using sub queries or direct queries. For example, ccrz.ccServiceProduct.getProducts(ccrz.ccAPI.SZ_SUBQUERY) will use sub queries to fetch the related entities for each product, while ccrz.ccServiceProduct.getProducts(ccrz.ccAPI.SZ_DIRECTQUERY) will use direct queries to fetch the related entities separately.
Object type casting: This operation indicates that the data should be cast to a specific object type. For example, ccrz.ccServiceProduct.getProducts(ccrz.ccAPI.SZ_SOBJECT) will cast the data to sObjects instead of transformed objects.
Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Data Sizing Conventions

Question: 26

A user wants the pricing to reflect the price values stored in an external ERP during the checkoutflow. In what way can this requirement be satisfied?

A. Override the computePricingCart method in ccrz.cc_api_PriceAdjustoment and make the callout in this method.
B. None of the above
C. Override the computePricingReview method in ccrz.cc_CartExtension and make the callout in this method.
D. Override the computePricingCart methos in ccrz.cc_api_CartExtension and make the callout in this method.

Answer: D

Explanation:

To reflect the price values stored in an external ERP during the checkout flow, the requirement can be satisfied by overriding the computePricingCart method in ccrz.cc_api_CartExtension and making the callout in this method. This method is responsible for computing the pricing for the cart and its line items. By overriding this method, the user can make a callout to the external ERP and update the pricing information accordingly. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Logic Classes, ccLogicCart Class

Question: 27

A user wants to have a Contact Us page in the storefront. This page will be a web-tolead form and it should have the header and footer, essentially the same look and feel as all the pages in the application. How can this requirement be fulfilled?

A. Page Include
B. Subscriber Page (CC Page)
C. Subscriber Template
D. Body Include Begin

Answer: B

Explanation:

To have a Contact Us page in the storefront that is a web-to-lead form and has the same look and feel as all the pages in the application, the requirement can be fulfilled by creating a Subscriber Page (CC Page). This is a custom Visualforce page that can be added to the storefront and use the standard header and footer components. The page can also include a web-to-lead form that submits data to Salesforce as leads. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Subscriber Pages

Question: 28

A user wants to have a customized experience for adding items to the cart. The user also wants the mini cart module to reflect changes to the state of the cart afterwords. How should this requirement be fulfilled?

A. Leverage the Addto Cart Global API which add items to the cart and also refreshes the page with the new data.
B. Trigger the global „cartChange" event and then trigger „changeMiniCart" event after the Add to Cart Action on the custom button.
C. Write a custom Remote Action to refresh the Mini Cart and refresh the Cart Line item count on the Cart Link in the header.
D. Trigger the global „cartChange" event after the Add to Cart Action on the custom button.

Answer: D

Explanation:

To have a customized experience for adding items to the cart and also update the mini cart module, the requirement can be fulfilled by triggering the global “cartChange” event after the Add to Cart Action on the custom button. This event will notify all the components that are listening to it that the cart has changed and they should refresh their data accordingly. The mini cart module is one of these components, so it will update its state based on the new cart data. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Events

Question: 29

A user wants to leverage a three columnlayout on a page. The user also wants to move the mini-cart widget from the right to the center column. How can this requirement be fulfilled?

A. Gross Layout Override

B. Subscriber Template
C. Page Include
D. HandleBar Template Override

Answer: A

Explanation:

To leverage a three column layout on a page and move the mini-cart widget from the right to the center column, the requirement can be fulfilled by creating a Gross Layout Override. This is a custom Visualforce page that overrides the default layout of a page and allows changing its structure and content. The user can create a Gross Layout Override that uses a three column layout and places the mini-cart widget in the center column. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Gross Layout Overrides

Question: 30

What are the templating, Javascript, and CSS frameworks what the cloudcraze managed package leverages?

A. Angularjs, Backbonejs, and handlebarsjs
B. Bootstrap, Backbonejs, and handlebarsjs
C. Bootstrap, Angularjs, and Backbonejs
D. Angularjs, react.js, and handlebarsjs

Answer: B

Explanation:

The templating, JavaScript, and CSS frameworks that the cloudcraze managed package leverages are Bootstrap, Backbone.js, and Handlebars.js. Bootstrap is a CSS framework that provides responsive design and layout components. Backbone.js is a JavaScript framework that provides models, views, collections, and events for building single-page applications. Handlebars.js is a templating engine that allows generating HTML from JSON data. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Front-End Development

Question: 31

What are three advantages of using ccLog over the Salesforce standard System.debug class? (3 answers)

A. There is no need to use string concatenation to easily tag log statements with a subject.
B. ccLog can debug syntax errors found in the JavaScript.
C. There is no need to create a User Trace Flag.
D. Append #ccLog=<Logging Token Name> to the end of the storefront URL in order to get logs in the

inspector console.
E. There is no need to manually set a cookie to debug with the Site Guest User.

Answer: A,D,E

Explanation:

Three advantages of using ccLog over the Salesforce standard System.debug class are:

There is no need to use string concatenation to easily tag log statements with a subject. ccLog allows passing a subject parameter to the log method, which will prepend the subject to the log message. For example, ccLog.log('This is a message', 'Subject') will log [Subject] This is a message.
There is no need to create a User Trace Flag. ccLog can be enabled by setting the value of CO.logToken to true in CCAdmin, which will activate logging for all users who access the storefront. There is no need to manually set a cookie to debug with the Site Guest User. ccLog can be enabled for the Site Guest User by appending #ccLog=<Logging Token Name> to the end of the storefront URL in order to get logs in the inspector console. For example, https://my- storefront.com/#ccLog=debug will enable logging for the Site Guest User with the debug level. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Logging

Question: 32

What are three ways to implement custom post Order processing? (3 answers)

A. Use a Salesforce workflow rule that executes when an Order record is created.
B. Extend cc_hk_invoice tohandle custom business logic post Order processing
C. Use cc_hk_Order.placeTarget to define a new Order Confirmation page which executes additional business logic.
D. Modify or add custom Cart formula fields to handle logic.
E. Use Process builder to implement business processes that execute when an Order record is created.

Answer: B,C,E

Explanation:

Three ways to implement custom post Order processing are:
Extend cc_hk_invoice to handle custom business logic post Order processing. This hook allows modifying the invoice data or performing additional actions after an Order is placed. For example, the hook can send an email notification or update a custom field on the invoice record.
Use cc_hk_Order.placeTarget to define a new Order Confirmation page which executes additional business logic. This hook allows specifying a custom Visualforce page that will be displayed after an Order is placed. The custom page can include additional business logic or user interface elements. Use Process Builder to implement business processes that execute when an Order record is created. Process Builder is a tool that allows creating workflows and actions based on criteria and conditions. For example, Process Builder can create a task, send an email, or update a record when an Order record is created. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Hooks, Process Builder

Question: 33

What are three ways to test the value of Page Label on any Salesforce B2B Commerce Community Page? (3 answers)

A. Access the source HTML for the page viathe browser developer tools.
B. Execute CCRZ.pagevars.pageLabels['PAGE_LABEL_NAME') in the JavaScript console.
C. Execute CCRZ.processPageLabelMap('PAGE_LABEL_NAME') in the JavaScript console.
D. Enable the 'display page label names' in cc admin.
E. Execute (('PAGE_LABEL_NAME')) in the JavaScript console

Answer: B,C,D

Explanation:

Three ways to test the value of Page Label on any Salesforce B2B Commerce Community Page are: Execute CCRZ.pagevars.pageLabels['PAGE_LABEL_NAME'] in the JavaScript console. This will return the value of the page label with the given name from the pagevars object, which contains all the page labels that are used on the page. Execute CCRZ.processPageLabelMap('PAGE_LABEL_NAME') in the JavaScript console. This will return the value of the page label with the given name from the pageLabelMap object, which contains all the page labels that are defined in CCAdmin.
Enable the 'display page label names' in cc admin. This will display the name of each page label next to its value on the storefront pages, which can help identify and verify the page labels. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Page Labels

Question: 34

What are two guidelines for logging that are used within the core Salesforce B2B Commerce product? (2 answers)

A. Items or data within computational intensive loops shouldbe logged.
B. The close method of ccrz.ccLog must be called at the end of the remote action.
C. No calls to ccrz.ccLog can be made before cc_CallContext.initRemoteContext is executed.
D. It is okay to log any data on the server that is already logged on the client side.

Answer: B,C

Explanation:

Two guidelines for logging that are used within the core Salesforce B2B Commerce product are:
The close method of ccrz.ccLog must be called at the end of the remote action. This method will flush the log messages to the browser console or to a custom object, depending on the logging mode. If this method is not called, the log messages may not be displayed or saved properly.
No calls to ccrz.ccLog can be made before cc_CallContext.initRemoteContext is executed. This method will initialize the call context object, which contains information such as the current user, cart, storefront, and configuration settings. These information are required for logging, so calling ccrz.ccLog before initializing the call context will result in an error.
Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Logging

Question: 35

What is a best practice when passing query parameters from user interface to an apex controller?

A. Query parameters should be properly sanitized by using JSINHTMLENCODE within the VisualForce Page or Component.
B. String parameters should be trimmed using String.trim().
C. Query parameters should be passed only to Salesforce B2B Commerce classes that you are extending.
D. Query parameters should be stored on a backbone model prior to passing them to the server

Answer: A

Explanation:

A best practice when passing query parameters from user interface to an apex controller is to query parameters should be properly sanitized by using JSINHTMLENCODE within the VisualForce Page or Component. This function will encode any special characters in the query parameters to prevent cross-site scripting (XSS) attacks or SOQL injection

attacks. For example, ccrz.ccRemoteActions.getProducts('{!JSINHTMLENCODE(searchTerm)}') will encode the searchTerm parameter before passing it to the apex controller. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Security

Question: 36

What is a method to resolve if the current storefront customer is a Salesforce B2B Commerce guest user in an apex class?

A. ccrz.cc_CallContext.currUser.isGuest
B. ccrz.cc_CallContext.isGuest
C. UserInfo.getUserType()
D .. UserType

Answer: B

Explanation:

A method to resolve if the current storefront customer is a Salesforce B2B Commerce guest user in an apex class is to use ccrz.cc_CallContext.isGuest. This property will return true if the current user is a guest user, or false otherwise. For example, if(ccrz.cc_CallContext.isGuest){ // do something for guest user } will execute some logic only for guest users. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Call Context

Question: 37

What is a valid way of referencing the CC Cart Object whose API name is E_Cart c in a SOQL query?

A. _Cart c
B. c.E_Cart c

C. ccrz E_Cart c
D. cloudcraze E_Cart c

Answer: C

Explanation:

A valid way of referencing the CC Cart Object whose API name is E_Cart c in a SOQL query is to use ccrz E_Cart c. This is the transformed name of the object that is used by the Salesforce B2B Commerce framework. All custom objects and fields that are part of the cloudcraze managed package have the prefix ccrz in their API names. For example, SELECT Id, Name FROM ccrz E_Cart c will query the CC Cart Object records. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Query Transformation

Question: 38

What is a valid way of referencing the global cc_api_CartExtentsion apex class via subscriber code?

A. ccrz cc_api_CartExtension

B. c cc_api_CartExtension
C. cloudcraze.cc_api_CartExtension
D. ccrz.cc_api_CartExtension

Answer: D

Explanation:

A valid way of referencing the global cc_api_CartExtension apex class via subscriber code is to use ccrz.cc_api_CartExtension. This is the name of the class that is defined in the cloudcraze managed package. The class is global, so it can be accessed by subscriber code. The other options are either invalid or incorrect. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, API Classes

Question: 39

What is default behavior for how theSalesforce B2B Commerce Global APIs transform Salesforce data?

A. Fields names are returned using the Salesforce naming convention.
B. Fields names are returned with „c." prepended in their name.
C. Fields names are returned with a lowercase first letter,camelcase convention
D. Fields names can be mapped to any naming convention desired

Answer: C

Explanation:

The default behavior for how the Salesforce B2B Commerce Global APIs transform Salesforce data is to return field names with a lowercase first letter, camelcase convention. For example, the field name

ccrz E_Product c in Salesforce will be transformed to eProduct in the API. This is done to follow the JavaScript naming convention and to avoid conflicts with the standard Salesforce fields and relationships. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Query Transformation

Question: 40

What is essential for a Salesforce B2B Commerce theme to show up in the theme section in CC Admin?

A. The theme needsto be set as a Custom Setting in Salesforce.
B. The theme needsto be set in the Configuration Settings.
C. The theme needsto have "theme" in the name of the Static Resource.
D. The theme needsto be referred to in the head element on the page

Answer: C

Explanation:

An essential requirement for a Salesforce B2B Commerce theme to show up in the theme section in CC Admin is that the theme needs to have “theme” in the name of the Static Resource. For example, a theme named “MyTheme” will not appear in CC Admin, but a theme named “MyTheme_theme” will. This is how the framework identifies which static resources are themes and which are not.

Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Themes

Question: 41

What isthe difference between Gross Layout Overrides and Subscriber Templates?

A. Gross Layout Overrides allow modification to CSS of a page, while Subscriber Templates allows for modification of the entire page including header and footer.
B. Subscriber Templates allows for modification of the header, the footer and the content in between them. Gross Layout Overrides only allow for modification of the header and footer.
C. Subscriber Templates allow for modification of the header and the footer, while Gross Layout Overrides allow for modification everything inside the header and footer.
D. Gross Layout Overrides allow for the modification of the footer, while Subscriber Templates allow for modification of everything inside the header and footer.

Answer: C

Explanation:

The difference between Gross Layout Overrides and Subscriber Templates is that Subscriber Templates allow for modification of the header and the footer, while Gross Layout Overrides allow for modification everything inside the header and footer. Subscriber Templates are custom Visualforce pages that can replace the standard header and footer components on a page. Gross Layout Overrides are custom Visualforce pages that can replace the default layout of a page, such as the number of columns, the position of widgets, or the content of sections. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Subscriber Templates, Gross Layout Overrides

Question: 42

What is the recommended method for increasing the number of required autocomplete characters that are typed before autocomplete works?

A. Override and extend the autoComplete method in cc_hk_Catalog.
B. Override theautoComplete.search_input.minLength value in the CCRZ.uiProperties file
C. Override the autocomplete template and create and register a new handlebars helper.
D. Update the...minLength property in CC Admin, then build and activate a new cache.

Answer: B

Explanation:

The recommended method for increasing the number of required autocomplete characters that are typed before autocomplete works is to override the autoComplete.search_input.minLength value in the CCRZ.uiProperties file. This file contains various properties that control the behavior and appearance of the user interface components. The autoComplete.search_input.minLength property specifies the minimum number of characters that must be entered before the autocomplete feature is triggered. The default value is 3, but it can be changed to any desired value by overriding it in the CCRZ.uiProperties file. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, UI Properties

Question: 43

What is true regarding adding more Configuration Settings to Salesforce B2B Commerce?

A. Metadata can be added to existing modules, but you cannot add new modules.
B. Configuration settings can only be extended through API's
C. Select "New"in your storefront's Configuration Settings and create a custom setting.
D. More modules and metadata can be added to Salesforce B2B Commerce.

Answer: D

Explanation:

More modules and metadata can be added to Salesforce B2B Commerce by creating custom configuration settings. Configuration settings are custom settings that store various values and parameters that affect the functionality and appearance of the storefront. They are organized into modules, which group related settings together. To create a custom configuration setting, the user needs to create a custom setting record in Salesforce and specify its module, name, value, and description. The custom setting will then appear in CCAdmin under the specified module. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Configuration Settings

Question: 44

What two kinds of queries do the methods in Salesforce B2B Commerce services perform by default? (2 answers)

A. SOSL
B. SQL
C. SOQL
D. Schema-less queries

Answer: CD

Explanation:

Two kinds of queries that the methods in Salesforce B2B Commerce services perform by default are SOQL and schema-less queries. SOQL is the query language that is used to retrieve data from Salesforce objects and fields. Schema-less queries are queries that do not specify the object or field names explicitly, but use placeholders instead. For example, ccrz.ccServiceDao.getQuery('SELECT Id FROM Account WHERE Name = :name') is a schema-less query that uses :name as a placeholder for the field name. The framework will transform this query to use the actual field name based on the query transformation rules. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Query Transformation

Question: 45

When a user buys 10 units of product B, the user wants 1 unit of Product A to be automatically added to the cart. How can this requirement be fulfilled?

A. Override the AllowCheckout method in ccrz.cc_api_CartExtension
B. Override the prepareForSave method in ccrz.cc_api_CartExtension
C. Override the preprocess method in ccrz.cc_api_CartExtension

D. Override the prepareToAdd method in ccrz.cc_api_CartExtension

Answer: D

Explanation:

To automatically add 1 unit of Product A to the cart when a user buys 10 units of Product B, the requirement can be fulfilled by overriding the prepareToAdd method in ccrz.cc_api_CartExtension. This method is responsible for preparing the cart line items before they are added to the cart. By overriding this method, the user can check the quantity and product ID of the cart line items and add an additional cart line item for Product A if the condition is met. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, API Classes, cc_api_CartExtension Class

Question: 46

Where are handlebar Templates defined in the Cloudcraze managed package?

A. uiProperties file
B. Configuration Setting configuration
C. cc_hk_UserInterface extension
D. Page Settings configuration

Answer: A

Explanation:

Handlebar Templates are defined in the uiProperties file in the cloudcraze managed package. This file contains various properties that control the behavior and appearance of the user interface components. The handlebarTemplates property defines a map of template names and template contents that are used by the handlebars.js templating engine to generate HTML from JSON data.
Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, UI Properties, Handlebars Templates

Question: 47

Where is the API-based record creation generally handled in Salesforce B2B Commerce?

A. In the methods available in extension hooks
B. The service-layer responsible for the entity
C. Data creation is not allowed
D. Logic classes that implement the businesslogic for create operations

Answer: B

Explanation:

The API-based record creation is generally handled in the service-layer responsible for the entity in Salesforce B2B Commerce. The service-layer is a set of classes that provide methods for interacting with the data layer and performing business logic. Each entity, such as product, cart, or order, has a corresponding service class that handles the create, read, update, and delete operations for that entity. For example, ccrz.ccServiceProduct provides methods for creating and retrieving products. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Service Classes

Question: 48

Which cookie stores the effective account ID when effective account is enabled?

A. apex cclgtkn
B. apex effacc
C. apex currCartId
D. apex cc_anonymous_Country

Answer: B

Explanation:

The cookie that stores the effective account ID when effective account is enabled is apex effacc. This cookie is set by the cc_hk_EffectiveAccount hook when a user switches to another account that they have access to. The cookie value is the ID of the effective account, which is used to determine

the pricing, availability, and visibility of products and categories for that account. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Effective Account

Question: 49

Which event is invoked by any CCRZ Salesforce B2B CommeceView after the view is rendered?

A. view:*:load
B. view:*:refresh
C. view:*:onload
D. view:*:rendered

Answer: D

Explanation:

The event that is invoked by any CCRZ Salesforce B2B Commerce View after the view is rendered is view:*:rendered. This event is triggered by the render method of the CCRZ.View class, which is the base class for all views in the framework. The event can be used to perform any actions or logic that depend on the view being rendered, such as initializing widgets, binding events, or updating the user interface. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Events, View Class

Question: 50

Which event is triggered within Salesforce B2B Commerce whenever a cart's statechanges?

A. cartChange
B. cart
C. pageMessage
D. cartState

Answer: A

Explanation:

The event that is triggered within Salesforce B2B Commerce whenever a cart's state changes is cartChange. This event is triggered by the updateCart method of the CCRZ.Cart class, which is responsible for updating the cart data and performing any actions that depend on the cart state, such as applying discounts, taxes, or shipping charges. The event can be used to notify all the components that are listening to it that the cart has changed and they should refresh their data accordingly. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Events, Cart Class

Question: 51

Which event should be triggered when user facing info, warning or error messages need to be displayed on a Visualforce page?

A. showMessage

B. displayPageMessage
C. displayMessage
D. pageMessage

Answer: D

Explanation:

The event that should be triggered when user facing info, warning or error messages need to be displayed on a Visualforce page is pageMessage. This event is triggered by the displayPageMessage method of the CCRZ.util class, which is a utility class that provides various helper methods for common tasks. The event can be used to display a message on the page with a specified type, title, and text. For example, CCRZ.util.displayPageMessage('error', 'Oops!', 'Something went wrong.') will trigger the pageMessage event and display an error message on the page. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Events, Util Class

Question: 52

Which format is the custom Salesforce field with the API name
"My_Fiels_Name c" transformed onto by default in Salesforce B2B Commerce?

A. MyFieldName
B. myFieldName
C. myfieldname
D. My_Field_Name c

Answer: B

Explanation:

The format that the custom Salesforce field with the API name My_Field_Name c is transformed onto by default in Salesforce B2B Commerce is myFieldName. This is done to follow the JavaScript naming convention and to avoid conflicts with the standard Salesforce fields and relationships. The transformation rules are as follows:
Remove any namespace prefix from the field name, such as ccrz or cloudcraze .

Remove any underscores from the field name and capitalize the first letter of each word after an underscore, such as MyFieldName.
Lowercase the first letter of the field name, such as myFieldName. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Query Transformation

Question: 53

Which format is the custom Salesforce relationship with the API name, "My_Relationship_Name r.My_Name c" queried and transformed into dy default in Salesforce B2B Commerce?

A. myrelationshipname.myname: value
B. myRelationshipName.myName: value
C. myRelationshipNameR=>(myName: value)
D. My_Relationship_Name r.My_Name c: value

Answer: C

Explanation:

The format that the custom Salesforce relationship with the API name, My_Relationship_Name r.My_Name c is queried and transformed into by default in Salesforce B2B Commerce is myRelationshipName.myName: value. This is done to follow the JavaScript naming convention and to avoid conflicts with the standard Salesforce fields and relationships. The transformation rules are as follows:
Remove any namespace prefix from the field name, such as ccrz or cloudcraze .
Remove any underscores from the field name and capitalize the first letter of each word after an underscore, such as MyRelationshipName and MyName.
Lowercase the first letter of the field name, such as myRelationshipName and myName.
Use a dot (.) to separate the relationship name and the field name, such as myRelationshipName.myName.
Use a colon (:) to separate the field name and the field value, such as myRelationshipName.myName: value. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Query Transformation

Question: 54

Which Global JavaScript Object should be extended when writing custom Remote Actions?

A. CCRZ.
B. CCRZ.cc
C. CCRZ.cc_CallContext
D. CCRZ.RemoteInvocation

Answer: B

Explanation:

The Global JavaScript Object that should be extended when writing custom Remote Actions is CCRZ.cc. This object contains all the Remote Actions that are defined in the cloudcraze managed package, which can be overridden or extended by

subscriber code. The object also provides a mechanism for registering custom Remote Actions that can be invoked by the user interface components. For example, CCRZ.cc.customAction = function(params, callback){ // do something } will define a custom Remote Action named customAction that can be called by CCRZ.cc.customAction(params, callback). Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Remote Actions

Question: 55

Which handlebars helper expression is used in Salesforce B2B Commerce pages and components to toggle the display of a block of markup?

A. {{#ifStoreSetting 'Field c'}} ... {{/ifStoreSetting}}
B. {{#ifSetting 'Page.cfg}} ... {{/ifSetting}}
C. {{#ifConfig 'Field c'}} ... {{/ifConfig}}
D. {{#ifDisplay 'Page.cfg'}} ... {{/ifDisplay}}

Answer: C

Explanation:

The handlebars helper expression that is used in Salesforce B2B Commerce pages and components to toggle the display of a block of markup is {{#ifConfig 'Field c'}} ... {{/ifConfig}}. This expression will evaluate the value of the configuration setting with the API name Field c and render the block of markup if the value is true, or skip it if the value is false. For example, {{#ifConfig 'CO.showMiniCart'}} ... {{/ifConfig}} will render the mini-cart div only if the configuration setting CO.showMiniCart is true. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Handlebars Helpers

Question: 56

Which handlebars helper function is used on Salesforce B2B Commerce pages and components for formatting price values?

A. formatPrice
B. priceAbs
C. showprice
D. price

Answer: A

Explanation:

The handlebars helper function that is used on Salesforce B2B Commerce pages and components for formatting price values is formatPrice. This function will format a numeric value as a price according to the currency settings and locale of the storefront. For example, {{formatPrice price}} will format the price value as $1,234.56 for US dollars or €1.234,56 for euros. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Handlebars Helpers

Question: 57

Which method is used to override when extending the Salesforce B2B Commerce logic providers?

A. doLogic
B. process
C. doAction
D. fetch

Answer: C

Explanation:

The method that is used to override when extending the Salesforce B2B Commerce logic providers is doLogic. This method is responsible for executing the business logic for each logic provider class, such as validating inputs, performing calculations, applying rules, or updating data. By overriding this method, the user can modify or extend the existing logic or add their own custom logic. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Logic Classes

Question: 58

Which method needs to be implemented when rendering a Salesforce B2B Commerce view in order to have it called after rendering has finished?

A. There are no methods called on the view after rendering has finished
B. onRender()
C. postRender()
D. afterRender()

Answer: C

Explanation:

The method that needs to be implemented when rendering a Salesforce B2B Commerce view in order to have it called after rendering has finished is postRender. This method is an optional hook that can be defined by the user to perform any actions or logic that depend on the view being rendered, such as initializing widgets, binding events, or updating the user interface. The method will be called by the render method of the CCRZ.View class, which is the base class for all views in the framework. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, View Class

Question: 59

Which method signature is used in the Global API's?

A. Changes based on API and Method name
B. ccrz.cc_Output (ccrz:cc_Input input)
C. Map<String, Object>
D. List<List<Object>>

Answer: B

Explanation:

The method signature that is used in the Global API's is ccrz.cc_Output (ccrz.cc_Input input). This signature indicates that the Global API methods take a single input parameter of type ccrz.cc_Input and return an output of type ccrz.cc_Output.

These are custom classes that are defined in the cloudcraze managed package and contain various properties and methods for handling the input and output data. For example, ccrz.ccServiceProduct.getProducts(ccrz.cc_Input input) is a Global API method that takes an input object and returns an output object containing product data. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, API Classes, cc_Input Class, cc_Output Class

Question: 60

Which out of the box Salesforce B2B Commerce page can give instructions to web crawlers from accessing specific Salesforce B2B Commerce pages?

A. CCCat?SiteMap
B. cc_RobotsTxT
C. CCSiteIndex
D. CCPage

Answer: B

Explanation:

The out of the box Salesforce B2B Commerce page that can give instructions to web crawlers from accessing specific Salesforce B2B Commerce pages is cc_RobotsTxt. This is a Visualforce page that generates a robots.txt file, which is a text file that tells web crawlers which pages or files they can or can't request from a site. The page uses the configuration settings CO.RobotsTxtAllow and CO.RobotsTxtDisallow to specify which paths are allowed or disallowed for web crawlers. For example, User-agent: * Disallow: /CCCart will instruct web crawlers to not access the CCCart page. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Robots.txt File

Question: 61

Which Salesforce B2BCommerce object needs to have a record added when defining a new Subscriber Pages to be rendered in a CC Page?

A. CC Storefront Assosiation
B. CC Admin
C. CC Page Sections
D. CC Subscriber Pages

Answer: D

Explanation:

The Salesforce B2B Commerce object that needs to have a record added when defining a new Subscriber Page to be rendered in a CC Page is CC Subscriber Pages. This is a custom object that stores information about the Subscriber Pages, such as the name, description, URL, and Visualforce page. To create a new Subscriber Page, the user needs to create a new record in this object and specify the required fields. The user can then select the Subscriber Page from the CC Page Settings configuration in CCAdmin. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Subscriber Pages

Question: 62

Which service method should be overridden in order to allow "without sharing" queries?

A. ccrz.ccService.queryWithoutSharing()

B. ccrz.ccAPI.queryService()
C. ccrz.ccService.query()
D. ccrz.ccService.initSVCDAO()

Answer: D

Explanation:

The service method that should be overridden in order to allow “without sharing” queries is ccrz.ccService.initSVCDAO. This method is responsible for initializing the service data access object (SVCDAO) that is used by the service class to perform queries. By overriding this method, the user can specify the sharing mode of the SVCDAO, which will determine whether the queries respect or ignore the sharing rules of the current user. For example, ccrz.ccService.initSVCDAO(ccrz.ccAPI.SZ_WITHOUTSHARING) will initialize the SVCDAO with the without sharing mode.
Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Service Classes, ccService Class

Question: 63

Which three actions are applicable when extendinga default Salesforce B2B Commerce page via a page include? (3 answers)

A. Create a Service Class override to query the new page include.
B. Create the VisualForce page you wish to include to the Salesforce b2B Commerce page.
C. Prepend "c." to the name ofthe page referenced in the configuration setting.
D. Create a configuration setting for enabling the page include and assigning the new page include via CC admin.
E. Build and activate a new configuration cache setting via CC admin.

Answer: B,D,E

Explanation:

Three actions that are applicable when extending a default Salesforce B2B Commerce page via a page include are:
Create the VisualForce page you wish to include to the Salesforce B2B Commerce page. This page will contain the custom content or logic that you want to add to the existing page. For example, you can create a VisualForce page named MyPageInclude that displays some additional information or functionality on the product detail page.
Create a configuration setting for enabling the page include and assigning the new page include via CC Admin. This setting will specify which page include you want to use for which page and section. For example, you can create a configuration setting named CO.MyPageInclude with the value PDV.BodyIncludeBegin.MyPageInclude, which means you want to use MyPageInclude as the body include begin for the product detail view (PDV) page.
Build and activate a new configuration cache setting via CC Admin. This action will refresh the cache and apply the changes to the storefront. You need to do this whenever you make any changes to the configuration settings or VisualForce pages.
Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Page Includes

Question: 64

Which three actions are applicable when modifying the number of steps required in the Salesforce Commerce Checkout flow? (3 answers)

A. Perform a template override on the Checkout page.
B. Add a page include to the checkout page.
C. Build and activate a new configuration cache setting via CC admin.
D. Set the value of the configuration setting defined as CO.useDef to TRUE
E. Set the value of the configuration setting defined as CO.overrideFlow to TRUE.

Answer: A,C,E

Explanation:

Three actions that are applicable when modifying the number of steps required in the Salesforce Commerce Checkout flow are:
Perform a template override on the Checkout page. This action will allow you to change the structure and content of the Checkout page, such as adding or removing sections, widgets, or fields. For example, you can override the checkout.handlebars template and modify it according to your requirements.
Set the value of the configuration setting defined as CO.overrideFlow to TRUE. This setting will enable you to use your own custom checkout flow instead of the default one. You need to set this value to true before you can modify the checkout flow.
Set the value of the configuration setting defined as CO.useDef to TRUE. This setting will enable you to use a single-page checkout flow instead of a multi-step checkout flow. You need to set this value to true if you want to reduce the number of steps in the checkout flow to one. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Checkout Flow

Question: 65

Which three attributes are true regarding Subscriber pages? (3 answers)

A. Subscriber pages can include additional standard Salesforce B2B Commerce components such as featured products, category tree, and mini cart.
B. All the user interface components must be createdmanually.
C. Subscriber pages allow customers to quickly create new custom pages for their storefront.
D. Out of the Box, Subscriber Pages are completely blank HTML pages.
E. Standard Salesforce B2B Commerce components are automatically included on the page e.g. Header links, images, menu items, containers, etc.

Answer: A,C,D

Explanation:

Three attributes that are true regarding Subscriber Pages are:
Subscriber Pages can include additional standard Salesforce B2B Commerce components such as featured products, category tree, and mini cart. These components can be added to the Subscriber Pages by using the CCRZ.pagevars.pageLabels and CCRZ.pagevars.pageSections objects, which

contain the page labels and page sections that are defined in CCAdmin.
Subscriber Pages allow customers to quickly create new custom pages for their storefront. These pages can be used to display any content or functionality that is not available in the default pages, such as FAQs, testimonials, or promotions.
Out of the Box, Subscriber Pages are completely blank HTML pages. These pages do not have any predefined layout or content, so the user can customize them according to their requirements. The user can use Visualforce, HTML, CSS, JavaScript, or any other web technologies to create their own Subscriber Pages. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Subscriber Pages

Question: 66

Which three pages should be enabled for the Guest user profile for a storefront to have anonymous checkout? (3 answers)

A. CCPaymentInfo
B. CheckoutNew
C. OrderView
D. Checkout
E. OrderConfirmation

Answer: A,B,E

Explanation:

Three pages that should be enabled for the Guest user profile for a storefront to have anonymous checkout are:
CCPaymentInfo: This page allows the guest user to enter their payment information, such as credit card number, expiration date, and security code. The page also displays the order summary and total amount.
CheckoutNew: This page allows the guest user to enter their shipping and billing information, such as name, address, phone number, and email. The page also displays the cart items and shipping options.
OrderConfirmation: This page displays the confirmation message and order number after the guest user places their order. The page also provides a link to view the order details or print the invoice.
Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Anonymous Checkout

Question: 67

Which three statements are true about Global API versioning? (3 answers)

A. Calling in with an API version set to lower than 1 will result in an exceptional case where the exception classccrz.BelowMinAPIVersionException will be returned tocallers.
B. There is no need to pass API_VERSION to the Global APIs, and based on the Salesforce B2B Commerce Managed Package version, Global APIs are able to figure out what version of the API to use.
C. The API version is scoped at the Class API level and NOT at the method level.
D. Minimum API_VERSION is 1 and the Maximum API version follows the releases. E.g. The maximum was 4 as of Salesforce B2B Commerce Release-4.5, 5 as of Salesforce B2B CommerceRelease 4.6, etc.
E. Calling in with an API version set to more than current maximum will result in exception case where the exception class ccrz.ExceedsMaxAPIVersionException will be returned to callers.

Answer: A,D,E

Explanation:

Three statements that are true about Global API versioning are:
Calling in with an API version set to lower than 1 will result in an exceptional case where the exception class ccrz.BelowMinAPIVersionException will be returned to callers. This exception indicates that the API version is not supported by the framework and the caller should use a higher API version.
The API version is scoped at the Class API level and NOT at the method level. This means that all the methods in a class will use the same API version that is specified by the caller. For example, if the caller passes an API version of 4 to ccrz.ccServiceProduct.getProducts(), then all the other methods in ccrz.ccServiceProduct will also use API version 4.
Calling in with an API version set to more than current maximum will result in exception case where the exception class ccrz.ExceedsMaxAPIVersionException will be returned to callers. This exception indicates that the API version is not supported by the framework and the caller should use a lower API version. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, API Versioning

Question: 68

Which three statements are true regarding event handling in the Salesforce B2B Commerce managed package? (3 answers)

A. Salesforce B2B Commerce relies on a series of Javascript click listener events.
B. Writing directly to your own custom Backbone JS Views and Models automatically integrates that data into the Salesforce B2B Commerce Backbone Views and Models.
C. Salesforce B2B Commerce relies on a central event channel for communication across various Salesforce B2B Commerce Backbone Views and Models called CCRZ.pubSub.
D. CCRZ.pubSub defines the following three functions which can be used for event handling: trigger, on, and once.
E. CCRZ.pubSub extends the Backbone.Events JavaScript object.

Answer: C,D,E

Explanation:

Three statements that are true regarding event handling in the Salesforce B2B Commerce managed package are:
Salesforce B2B Commerce relies on a central event channel for communication across various Salesforce B2B Commerce Backbone Views and Models called CCRZ.pubSub. This object allows different components to publish or subscribe to events without being coupled with each other. For example, CCRZ.pubSub.trigger('cartChange') will publish an event named cartChange that can be subscribed by any component that needs to react to it.
CCRZ.pubSub defines the following three functions which can be used for event handling: trigger, on, and once. The trigger function is used to publish an event with a name and optional arguments. The on function is used to subscribe to an event with a name and a callback function. The once function is similar to on, but it will only execute the callback function once for each event.
CCRZ.pubSub extends the Backbone.Events JavaScript object. This object provides methods for managing events in Backbone.js, which is a JavaScript framework that provides models, views, collections, and events for building single-page applications. CCRZ.pubSub inherits all the methods and properties of Backbone.Events and adds some custom ones.
Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Events, PubSub Class

Question: 69

Which three steps are necessary to have subscriberpage added to Salesforce B2B Commerce after creating a custom Visualforce page? (3 answers)

A. Create a new CC Subscriber Page record that points to your custom Visualforce page.
B. Create a new Visualforce page, and manually import the Salesforce B2BCommerce JavaScript libraries. Run in Anonymous Apex ccrz.cc_util_Reflection.upsertPageUIKey([arg1],[arg2],[arg3]);
C. Refresh the Page Keys Index in CC Admin.
D. Enable the Subscriber Page in CC Admin.

Answer: A,C,D

Explanation:

Three steps that are necessary to have a subscriber page added to Salesforce B2B Commerce after creating a custom Visualforce page are:
Create a new CC Subscriber Page record that points to your custom Visualforce page. This record will store information about the subscriber page, such as the name, description, URL, and Visualforce page. For example, you can create a new record named MySubscriberPage that points to your custom Visualforce page named MyPage.
Refresh the Page Keys Index in CC Admin. This action will update the page keys index, which is a cache that stores the mapping between the page keys and the subscriber pages. You need to do this whenever you create or modify a subscriber page record.
Enable the Subscriber Page in CC Admin. This action will allow you to select the subscriber page from the CC Page Settings configuration and assign it to a CC Page. For example, you can enable MySubscriberPage and assign it to the Home page.
Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Subscriber Pages

Question: 70

Which two aspects are applicable to Page Includes? (2 answers)

A. Standard Visualforce controls such as apex:form should not be used within a page include
B. Page Includes must be assigned to an OOTB Page, i.e. Home, Product Detail, etc., and enabled
C. Page Includes can be configured as Body Includes Begin.
D. If a controller is used for an included page, then a merge variable must be present on the page.

Answer: A,C

Explanation:

Two aspects that are applicable to Page Includes are:
Standard Visualforce controls such as apex:form should not be used within a page include. This is because the page include is rendered inside an existing Visualforce page that already has a form element. Using another form element inside the page include will cause conflicts and errors. Instead, the page include should use HTML elements or custom components that do not require a form element.
Page Includes can be configured as Body Includes Begin. This means that the page include will be rendered at the beginning of the body section of the page, before any other content or widgets. This can be useful for adding some custom content or functionality that applies to the whole page, such as a banner, a modal, or a script. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Page Includes

Question: 71

Which two different types of services do Salesforce B2B

A. Commerce APIs leverage? (2 answers) A) Logic services which are responsible for implementing business logic associated with entities.
B. Global services which are responsible for querying and transforming data from entities
C. Data services which are responsible for querying and transforming data from entities
D. Hook services which are extension points for subscribers to tie into.

Answer: A,C

Explanation:

Two different types of services that Salesforce B2B Commerce APIs leverage are:
Logic services which are responsible for implementing business logic associated with entities. These services are defined in the logic classes, such as ccrz.ccLogicCart, ccrz.ccLogicOrder, or ccrz.ccLogicProduct. These classes provide methods for validating inputs, performing calculations, applying rules, or updating data for each entity.
Data services which are responsible for querying and transforming data from entities. These services are defined in the service classes, such as ccrz.ccServiceCart, ccrz.ccServiceOrder, or ccrz.ccServiceProduct. These classes provide methods for creating, reading, updating, and deleting data for each entity. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Service Classes, Logic Classes

Question: 72

Which two Salesforce B2B Commerce visualforce pages must be enabled at a Salesforce Community level to make the out of the box SEO functionality available? (2 answers)

A. CCSizeIndex
B. SizeMap
C. CCCatSiteMap
D. ProductMap

Answer: A,C

Explanation:

Two Salesforce B2B Commerce Visualforce pages that must be enabled at a Salesforce Community level to make the out of the box SEO functionality available are:
CCSizeIndex: This page generates a sitemap.xml file, which is a file that lists all the pages and resources on a site that can be crawled by web crawlers. The page uses the configuration settings CO.SiteMapIncludeProducts and CO.SiteMapIncludeCategories to specify which products and categories should be included in the sitemap.
CCCatSiteMap: This page generates a category sitemap file, which is a file that lists all the categories on a site that can be crawled by web crawlers. The page uses the configuration setting CO.SiteMapCategoryDepth to specify how many levels of subcategories should be included in the category sitemap. Salesforce Reference: B2B Commerce and D2C Commerce Developer Guide, Sitemap Files

Question: 73

Which two statement are true for Mass Order (2 answers)

A. Mass Order pricing is done via a batch job.
B. Mass order works with thedefault wishlists
C. The variation product is leveraged for SKUs.
D. Mass Order is mobile ready with the ccrz templates.

Answer: A,C

Explanation:

Mass Order pricing is done via a batch job, which means that the prices are not calculated in real time, but rather at a scheduled time. Mass order works with the variation product, which is a product that has multiple SKUs with different attributes, such as size or color. Mass order does not work with the default wishlists, which are used to store products that the customer wants to buy later. Mass order is not mobile ready with the ccrz templates, which are the default templates for the storefront pages. Salesforce Reference: B2B Commerce Developer Guide: Mass Order, B2B Commerce Developer Guide: Variation Products

Question: 74

Which two statements are true about Global API's in Salesforce B2B Commerce? (2 answers)

A. Each globalAPI method takes in a Map<String, Object> as a single parameter and returns a Map<String, Object>
B. Global APIs are versioned.
C. Global APIs will catch any Exceptions that are thrown as part of missing parameters.
D. Global APIs will handle Transaction scope, and the Subscriber calling it does not need to handle scope.

Answer: A,B

Explanation:

Global APIs are methods that can be called from any Apex class or Visualforce page within the B2B Commerce managed package or subscriber code. Each global API method takes in a Map<String, Object> as a single parameter and returns a Map<String, Object>. This allows for flexibility and extensibility of the API. Global APIs are versioned, which means that they have a version number in their name, such as ccApi_1_0. This ensures backward compatibility and allows for new features and enhancements to be added without breaking existing functionality. Global APIs do not catch any Exceptions that are thrown as part of missing parameters, which means that the caller is responsible for handling any errors or validations. Global APIs do not handle Transaction scope, and the Subscriber calling it does not need to handle scope. This means that the caller can decide whether to use transactions or not, depending on the business logic and performance requirements. Salesforce Reference: B2B Commerce Developer Guide: Global APIs, B2B Commerce Developer Guide: Global API Versioning

Question: 75

Which two statements are true regarding the cc_CallContext class in Salesforce B2B Commerce? (2 answers)

A. The Salesforce session is accessible via the getSession method

B. The class can be used internally within Salesforce B2B Commerce and in subscriber code to access context level parameters
C. The userLocale variable returns the current Locale for storefront.
D. The current storefront is accessible via thisclass

Answer: B,D

Explanation:

The cc_CallContext class is a utility class that provides access to various context level parameters, such as the current storefront, user, cart, price list, currency, locale, and session. The class can be used internally within Salesforce B2B Commerce and in subscriber code to access these parameters. The current storefront is accessible via this class by using the getStorefront method, which returns a cc_Storefront c object. The userLocale variable returns the current Locale for storefront, but it is not part of the cc_CallContext class. It is a global variable that can be accessed from any Visualforce page or component by using {!userLocale}. The Salesforce session is accessible via the getSession method, but it is not part of the cc_CallContext class either. It is a method of the cc_SessionUtil class, which is another utility class that provides methods for managing sessions. Salesforce Reference: [B2B Commerce Developer Guide: cc_CallContext Class], [B2B Commerce Developer Guide: cc_SessionUtil Class]

Question: 76

Which two steps are necessary to enable Salesforce B2B Commerce logging in the managed package?

A. Ensure you save a value in the Logging Token input field in the Global Settings section of CC Admin.
B. Turn On theCheckbox "Cloudcraze Logging" in CC Admin.
C. Ensure the value saved in the Logging token field is appended to the ccLog query parameter.
D. Set a cookie with the Id of the user accessing the storefront in CC Admin

Answer: A,C

Explanation:

To enable Salesforce B2B Commerce logging in the managed package, you need to do two steps. First, you need to save a value in the Logging Token input field in the Global Settings section of CC Admin. This value can be any string that you choose, such as “debug”. Second, you need to ensure that the value saved in the Logging token field is appended to the ccLog query parameter in the URL of the storefront page that you want to debug. For example, if your logging token is “debug”, then your URL should look like this: https://my-storefront.com/?ccLog=debug. This will enable logging for that page only. You do not need to turn on the checkbox “Cloudcraze Logging” in CC Admin, as this is an old setting that is no longer used. You also do not need to set a cookie with the Id of the user accessing the storefront in CC Admin, as this is not required for logging. Salesforce Reference: [B2B Commerce Developer Guide: Logging]

Question: 77

Which two usages of ccLog should be avoided in Salesforce B2B Commerce? (2 answers)

A. ccrz.ccLog.log(System.LoggingLevel.ERROR, 'D:something', myCallToGetMessage(pl)), where myCallToGetMessage(pl) is a method invocation
B. crz.ccLog.log(System.LoggingLevel.WARN, 'D:something', 'Something unexpected occurred: The data we were expecting for pl was not there,')

C. crz.ccLog.log(System.LoggingLevel.DEBUG, 'D:myOrderList', myOrderList), where myOrderList is a list of orders
D. crz.ccLog.log(System.LoggingLevel.DEBUG, 'D:myOrder', myOrder), where myOrder is an order object

Answer: A, C

Explanation:

You should avoid using ccLog in two scenarios. First, you should avoid using ccLog with System.LoggingLevel.ERROR when passing in a method invocation as the third parameter, such as ccrz.ccLog.log(System.LoggingLevel.ERROR, 'D:something', myCallToGetMessage(pl)). This is because if an exception occurs within the method invocation, it will not be caught by ccLog and will cause an unhandled exception on the page. Instead, you should use a try-catch block around the method invocation and log the exception separately. Second, you should avoid using ccLog with System.LoggingLevel.DEBUG when passing in a large collection or object as the third parameter, such as ccrz.ccLog.log(System.LoggingLevel.DEBUG, 'D:myOrderList', myOrderList), where myOrderList is a list of orders. This is because logging large objects can cause performance issues and consume a lot of heap space. Instead, you should use a loop to log only the relevant fields or properties of the

object or collection. You can use ccLog with System.LoggingLevel.WARN when passing in a string as the third parameter, such as ccrz.ccLog.log(System.LoggingLevel.WARN, 'D:something', 'Something unexpected occurred: The data we were expecting for pl was not there,'). This is a valid use case for logging a warning message. You can also use ccLog with System.LoggingLevel.DEBUG when passing in a small object as the third parameter, such as ccrz.ccLog.log(System.LoggingLevel.DEBUG, 'D:myOrder', myOrder), where myOrder is an order object. This is acceptable as long as the object is not too large or complex. Salesforce Reference: [B2B Commerce Developer Guide: Logging]

Question: 78

Why is the use of a standard Visualforce control such as apex:form discouraged in Salesforce B2B Commerce page includes and subscriber pages?

A. Visualforce "scopes" controls that are present on a page and scope of the control will be set to "ccrz"
B. Apex:form render DOM components slowly
C. The CCRZ Javascript object is not accessible within an apex:form control.
D. Javascript events are not supported within an apex:form control

Answer: C

Explanation:

The use of a standard Visualforce control such as apex:form is discouraged in Salesforce B2B Commerce page includes and subscriber pages because the CCRZ JavaScript object is not accessible within an apex:form control. The CCRZ JavaScript object is a global object that provides access to various B2B Commerce functions and properties, such as logging, utilities, events, and configuration. Using an apex:form control would prevent the developer from using the CCRZ JavaScript object and its features. Visualforce does not "scope" controls that are present on a page and scope of the control will be set to "ccrz", as this is not a valid statement. Apex:form does not render DOM components slowly, as this is not a performance issue. JavaScript events are supported within an apex:form control, as this is not a limitation. Salesforce Reference: B2B Commerce Developer Guide: CCRZ JavaScript Object, B2B Commerce Developer Guide: Page Includes

Question: 79

Witch static method invocation is used to initialize ccrz.cc_CallContext with information from ccrz.cc_RemoteActionContext and return an instance of ccrz.cc_RemoteActionResult in an apex @RemoteAction methos?

A. ccrz.cc_CallContext.init(ccrz.cc_RemoteActionContext)
B. ccrz.cc_CallContext.initCallContext(ccrz.cc_RemoteActionContext)
C. ccrz.cc_CallContext.initRemoteActionContext(ccrz.cc_RemoteActionContext)
D. ccrz.cc_CallContext.initializeCallContext(ccrz.cc_RemoteActionContext)

Answer: B

Explanation:

The static method invocation that is used to initialize ccrz.cc_CallContext with information from ccrz.cc_RemoteActionContext and return an instance of ccrz.cc_RemoteActionResult in an Apex @RemoteAction method is ccrz.cc_CallContext.initCallContext(ccrz.cc_RemoteActionContext). This method takes in a ccrz.cc_RemoteActionContext object as a parameter and returns a ccrz.cc_RemoteActionResult object. The ccrz.cc_RemoteActionContext object contains information about the current storefront, user, cart, price list, currency, locale, and session. The ccrz.cc_RemoteActionResult object contains information about the status, result, and errors of the remote action. The other methods are not valid or do not exist. Salesforce Reference: B2B Commerce Developer Guide: cc_CallContext Class, B2B Commerce Developer Guide: cc_RemoteActionContext Class, B2B Commerce Developer Guide: cc_RemoteActionResult Class

Question: 80

Which two scoped modules should a developer import in Lightning web components to check user permissions?

A. @salesforce/permission
B. @salesforce/customPermission
C. @salesforce/hasPermission
D. @salesforce/userPermission

Answer: AB

Explanation:

To check user permissions in Lightning web components, a developer should import two scoped modules: @salesforce/permission and @salesforce/customPermission. The @salesforce/permission module allows the developer to check if the user has access to a standard permission, such as View Setup or Modify All Data. The @salesforce/customPermission module allows the developer to check if the user has access to a custom permission, such as Enable Debug Mode or Manage Orders. The other modules do not exist or are not related to user permissions. Salesforce Reference: Lightning Web Components Developer Guide: Check User Permissions, Lightning Web Components Developer Guide: Import Salesforce Modules

Question: 81

A developer needs to create an event listener on a parent component programmatically. With the script below, what

should replace the text <EVENT LISTENER UNE>?

```
import {LightningElement) from lwc';
export default class Parent extends LightningElement { constructor!) {
   super!);
   <EVENTJJSTENER_LINE>

  handleNotification = () => {};
}
```

A. this.template.addEventListener(handleNotification);
B. this.template.addEventListener(this.handleNotification);
C. this.template.addEventListenerCnotification1, this.handleNotification);
D. addEventListener('notifJcatiorV, this.handleNotification);

Answer: C

Explanation:

To create an event listener on a parent component programmatically, the developer should use the following line of code:
this.template.addEventListener('notification', this.handleNotification);
This line of code adds an event listener to the template element of the parent component, which is the root element that contains all the child elements. The event listener listens for an event named 'notification', which is a custom event that can be dispatched by any child component. The event listener invokes a method named handleNotification, which is an arrow function defined in the parent component class. The handleNotification method receives the event object as a parameter and can perform any logic based on the event data. The other lines of code are either incorrect or incomplete.
Salesforce Reference: Lightning Web Components Developer Guide: Communicate with Events, Lightning Web Components Developer Guide: Create and Dispatch Events

Question: 82

A developer has the task to create custom Lightning web components (LWCs). Which two steps must a developer take when creating custom LWCs?

A. Create an Apex class.
B. Authorize an org for an SFDX project.
C. CloneaLWC.
D. Deploy a custom component.

Answer: B, D

Explanation:

To create custom Lightning web components (LWCs), a developer must take two steps: authorize an org for an SFDX project and deploy a custom component. Authorizing an org for an SFDX project
allows the developer to connect to a Salesforce org, such as a scratch org, a sandbox, or a production org, and use it as a

development environment. Deploying a custom component allows the developer to push the LWC code from the local project to the org and make it available for use. Creating an Apex class is not a required step for creating custom LWCs, as not all LWCs need to use Apex. Cloning an LWC is not a required step either, as it is an optional way to create a new LWC based on an existing one. Salesforce Reference: Lightning Web Components Developer Guide: Authorize an Org for Development, Lightning Web Components Developer Guide: Deploy Your Component

Question: 83

Which two event settings are required for a custom event called CustomEvent to fire from the Lightning web component and propagate up to the DOM?

A. bubbles: true
B. composed: true
C. cancelable: true
D. composed: false

Answer: A, B

Explanation:

To fire a custom event called CustomEvent from the Lightning web component and propagate it up to the DOM, the developer must set two event settings: bubbles and composed. The bubbles setting determines whether the event bubbles up through the component's ancestors in the DOM tree. The composed setting determines whether the event crosses the shadow boundary and reaches the light DOM. Setting both bubbles and composed to true allows the event to be handled by any element in the DOM that listens for it. The cancelable setting is not required for firing or propagating the event, as it only determines whether the event can be canceled by calling preventDefault() on it. Setting composed to false would prevent the event from reaching the light DOM and limit its propagation to the shadow DOM. Salesforce Reference: Lightning Web Components Developer Guide: Create and Dispatch Events, Lightning Web Components Developer Guide: Event Propagation

Question: 84

How can a developer establish communication between components that are not in the same DOM (Document Object Model) tree?

A. Use publish-subscribe pattern.
B. Configure targets property.
C. Use dispatch events.
D. Use @api decorators.

Answer: A

Explanation:

To establish communication between components that are not in the same DOM (Document Object Model) tree, a developer can use the publish-subscribe pattern. The publish-subscribe pattern is a

messaging pattern that allows components to communicate with each other without being directly connected or aware of each other. The components can publish events to a common channel and subscribe to events from that channel. The channel acts as a mediator that delivers the events to the subscribers. The developer can use a custom library or a

Salesforce platform service, such as Lightning Message Service or Platform Events, to implement the publish-subscribe pattern.

Configuring the targets property is not a way to communicate between components that are not in the same DOM tree, as it only defines where a component can be used in an app. Using dispatch events is not a way either, as it only works for components that are in the same DOM tree or have a parent-child relationship. Using @api decorators is not a way either, as it only exposes public properties or methods of a component to other components that use it. Salesforce

Reference: Lightning Web Components Developer Guide: Communicate Across Salesforce UI Technologies, Lightning Web Components Developer Guide: Communicate with Events, [Lightning Web Components Developer Guide: Communicate with Properties]

Question: 85

Which two guidelines should a developer consider when migrating aura components to LWC?

A. Migrate one component and then determine whether additional effort would make sense
B. Start with migrating trees of components (components within components)
C. Force all developers to write any new components using Lightning web components
D. Start with simple components that only render UI

Answer: A, D

Explanation:

When migrating aura components to LWC, a developer should consider two guidelines: migrate one component and then determine whether additional effort would make sense and start with simple components that only render UI. Migrating one component and then determining whether additional effort would make sense allows the developer to evaluate the benefits and costs of migration and decide whether to continue or stop. Migrating simple components that only render UI allows the developer to leverage the performance and modern features of LWC without much complexity or dependency on other components or services. Starting with migrating trees of components (components within components) is not a good guideline, as it can introduce more challenges and dependencies that can complicate the migration process. Forcing all developers to write any new components using Lightning web components is not a good guideline either, as it can create inconsistency and confusion among developers and users. Salesforce Reference: [Lightning Web Components Developer Guide: Migrate Aura Components to Lightning Web Components], [Lightning Web Components Developer Guide: Migration Considerations]

Question: 86

What are two considerations to keep in mind when including additional JavaScript files in a Lightning web component?

A. Each additional file needs a corresponding .js-meta.xml file.
B. The files must be ES6 modules and must have names that are unique within the component's

folder.

C. A module can export named functions or variables
D. Additional JavaScript files should be minified before deployment

Answer: B, C

Explanation:

When including additional JavaScript files in a Lightning web component, a developer should keep in mind two considerations: the files must be ES6 modules and must have names that are unique within the component's folder and a module can export named functions or variables. The files must be ES6 modules because LWC uses ES6 modules as the standard for modular JavaScript code. The files must have names that are unique within the component's folder because LWC uses the file name as the module identifier and does not allow duplicate identifiers. A module can export named functions or variables because LWC supports named exports, which allow a module to export multiple values with different names. Each additional file does not need a corresponding .js-meta.xml file, as this is only required for the main JavaScript file of the component. Additional JavaScript files should not be minified before deployment, as this is not necessary or recommended for LWC. Salesforce Reference: [Lightning Web Components Developer Guide: Include JavaScript Files], [Lightning Web Components Developer Guide: ES6 Modules]

Question: 87

Which wire adapter should a developer use to retrieve metadata about a specific object?

A. getObjectMetadata
B. getObjectInfo
C. getObject
D. getObjectDescribe

Answer: B

Explanation:

To retrieve metadata about a specific object, a developer should use the getObjectInfo wire adapter. The getObjectInfo wire adapter imports data from the @salesforce/schema module and returns an object that contains information such as the object's label, key prefix, fields, child relationships, record type infos, and theme. The getObjectMetadata wire adapter does not exist. The getObject wire adapter does not retrieve metadata, but rather returns a record object based on the record ID. The getObjectDescribe wire adapter does not exist either. Salesforce Reference: [Lightning Web Components Developer Guide: getObjectInfo], [Lightning Web Components Developer Guide: Import Salesforce Schema]

Question: 88

Which code statement should a developer use to import the ID of the current Lightning Experience

A. import id from '@salesforce/network/ld'
B. import id from '@salesforce/experience/ld'

C. import id from '@salesforce/site/ld'
D. import id from '@salesforce/community/ld'

Answer: D

Explanation:

To import the ID of the current Lightning Experience community, a developer should use the following code statement:
import id from '@salesforce/community/Id';
The @salesforce/community module allows the developer to access information about the current community, such as its ID, name, URL, and base path. The other modules do not exist or are not related to the community ID.

The @salesforce/network module is used to access information about the current network, such as its ID and name. The @salesforce/experience module is used to access information about the current user experience, such as whether it is standard or custom. The @salesforce/site module is used to access information about the current site, such as its name and prefix. Salesforce Reference: [Lightning Web Components Developer Guide: Import Community Information], [Lightning Web Components Developer Guide: Import Salesforce Modules]

Question: 89

Which wire adapter should a developer use to retrieve metadata about a specific picklist?

A. getPicklistMetadataValues
B. getPicklistMetadata
C. getPicklistValues
D. getPicklist

Answer: C

Explanation:

To retrieve metadata about a specific picklist, a developer should use the getPicklistValues wire adapter. The getPicklistValues wire adapter imports data from the @salesforce/ui-api module and returns an object that contains information such as the picklist's label, value, default value, validity, and controlling field values. The getPicklistMetadataValues wire adapter does not exist. The getPicklistMetadata wire adapter does not exist either. The getPicklist wire adapter does not exist either. Salesforce Reference: [Lightning Web Components Developer Guide: getPicklistValues], [Lightning Web Components Developer Guide: Import User Interface API]

Question: 90

Which category can receive signaling from a Lightning Message Channel?

A. only descendents (i.e. children)
B. only direct siblings
C. page (anywhere on it)
D. only ancestors (i.e. parents)

Answer: C

Explanation:

A Lightning Message Channel can receive signaling from any component on a page (anywhere on it). A Lightning Message Channel is a service that allows components to communicate with each other across different Salesforce UI technologies, such as Aura, LWC, Visualforce, and Lightning web apps. A component can subscribe to a message channel and receive events that are published by another component on the same or different page. The message channel acts as a mediator that delivers the events to the subscribers. A Lightning Message Channel does not receive signaling only from descendants (i.e., children), only from direct siblings, or only from ancestors (i.e., parents), as these are limitations of event-based communication. Salesforce Reference: [Lightning Web Components Developer Guide: Communicate Across Salesforce UI Technologies], [Lightning Web Components Developer Guide: Communicate with Events]

Question: 91

A developer is building a custom component in Lightning web components (LWC) that needs to fetch data from an API. Which lifecycle hook should the developer use to make the API call?

A. connectedCallback
B. renderedCallback
C. errorCallback
D. disconnectedCallback

Answer: A

Explanation:

To make an API call in a Lightning web component (LWC), a developer should use the connectedCallback lifecycle hook. The connectedCallback lifecycle hook is invoked when the component is inserted into the DOM. This is the ideal time to make an API call, as the component is ready to receive and display data. The developer can use the fetch API or a third-party library, such as axios, to make the API call and handle the response. The renderedCallback lifecycle hook is not a good choice for making an API call, as it is invoked every time the component is rendered or rerendered. This can cause unnecessary or repeated API calls and affect performance. The errorCallback lifecycle hook is not a good choice either, as it is invoked when an error occurs in the component or in one of its children. This is not related to making an API call, but rather to handling errors. The disconnectedCallback lifecycle hook is not a good choice either, as it is invoked when the component is removed from the DOM. This is not a suitable time to make an API call, as the component is no longer visible or active. Salesforce Reference: Lightning Web Components Developer Guide: Lifecycle Hooks, Lightning Web Components Developer Guide: Call an Apex Method Imperatively

Question: 92

A developer is creating a component to implement a custom Terms and Conditions checkbox at checkout in the Aura Commerce template.

Which method should the developer implement on the Lightning web component to ensure the user accepts the terms and conditions?

A. ComponentValidity
B. Validate
C. SaveCheckout
D. CheckValidity

Answer: B

Explanation:

To implement a custom Terms and Conditions checkbox at checkout in the Aura Commerce template, a developer should add a Desired Delivery Date input field during the checkout flow. The Desired Delivery Date input field allows the customer to enter a date when they want their order to be delivered. The developer can use the @api decorator to expose this field as a public property of the Lightning web component and bind it to the ccCheckoutOrder object. The developer can also use the @wire decorator to get the current cart object and use its properties, such as shipping

address and shipping method, to calculate and display an estimated delivery date based on the desired delivery date. The developer can also add validation logic to ensure that the desired delivery date is valid and acceptable. Adding the Expected Delivery Date field to the order confirmation email is not a good solution, as it does not allow the customer to choose or see their delivery date before placing their order. Displaying the Expected Delivery Date on the order page with a Lightning web component is not a good solution either, as it does not allow the customer to enter or change their delivery date after placing their order. Configuring an email alert to the customer when the Expected Delivery Date changes is not a good solution either, as it does not provide a consistent or reliable way of informing the customer about their delivery date. Salesforce Reference: B2B Commerce Developer Guide: Checkout Flow, B2B Commerce Developer Guide: Checkout Order Object, Lightning Web Components Developer Guide: Communicate with Properties

Question: 93

Northern Trail Outfitters (NTO) has a B2B Commerce store for its resellers. It has received many customer service calls involving questions about the delivery date of customer orders.
How should a developer expose delivery time estimates to NTO's customers in the storefront to reduce call volume?

A. Add the Expected Delivery Date field to the order confirmation email.
B. Add a Desired Delivery Date input field during the checkout flow.
C. Display the Expected Delivery Date on the order page with a Lightning web component.
D. Configure an email alert to the customer when the Expected Delivery Date changes.

Answer: C

Explanation:

To expose delivery time estimates to NTO's customers in the storefront, a developer should display the Expected Delivery Date on the order page with a Lightning web component. The Expected Delivery Date is a custom field on the Order object that stores the date when the order is expected to

be delivered to the customer. The developer can use the @wire decorator to get the current order object and use its properties, such as order number, status, total amount, and expected delivery date, to display them on the order page. The developer can also use Apex methods or third-party APIs to calculate and update the expected delivery date based on various factors, such as inventory availability, shipping method, shipping address, and carrier service level. Displaying the expected delivery date on the order page allows the customer to see their delivery time estimate at any time and reduce their need to call customer service. Adding the Expected Delivery Date field to the order confirmation email is not a good solution, as it does not allow the customer to see their delivery time estimate if they lose or delete their email. Adding a Desired Delivery Date input field during the checkout flow is not a good solution either, as it does not guarantee that the customer's desired delivery date will be met or reflect any changes in delivery time due to unforeseen circumstances. Configuring an email alert to the customer when the Expected Delivery Date changes is not a good solution either, as it can create confusion or frustration for the customer if they receive multiple or conflicting emails about their delivery date. Salesforce Reference: B2B Commerce Developer Guide: Order Object, [B2B Commerce Developer Guide: Order Page], [Lightning Web Components Developer Guide: Call an Apex Method Imperatively]

Question: 94

Which template will correctly display the details message only when areDetailsVisible becomes true given the

following code in a Lightning Web Component?

```
import { LightningElement) from lwc';
export default class HelloConditionalRendering extends LightningElement {
  areOetailsVisible = false;
  handleChange(event) {
      this.areDetailsVisible = event.target.checked;

I
```

A)

```
<template if:true=(areDetailsVisible)>
 <div class="slds-m-vertical_medium">
   These are the details!
 </div>
</template>
```

B)

```
template if:areDetailsVisible=(true)>
  <div class= slds-m-vertical_medium">
   These are the details!
  </div>
</template> _____
```

C)

```
<template if:true(areDetailsVisible)>
   <div class="slds-m-vertical_medium''>
    These are the details!
   </div>
</template>
```

D)

```
<template if;{areDetailsVisible}=true>
 <div class="slds-m-vertical_medium‘>
  These are the details!
 </div>
</template>
```

A. Option A
B. Option B

C. Option C

D. Option D

Answer: C

Explanation:

The template that will correctly display the details message only when areDetailsVisible becomes true given the following code in a Lightning Web Component is option C. Option C uses the if:true directive to conditionally render a template block based on the value of areDetailsVisible. If areDetailsVisible is true, the template block inside the <template if:true={areDetailsVisible}> tag will be rendered. Otherwise, it will be skipped. Option A is incorrect because it uses the if:false directive, which does the opposite of if:true. It renders the template block only when areDetailsVisible is false. Option B is incorrect because it uses an invalid syntax for the if directive. The if directive requires a colon (:) after the if keyword, not an equal sign (=). Option D is incorrect because it uses an invalid syntax for the template tag. The template tag requires a closing tag (</template>), not a self-closing tag (<template/>). Salesforce Reference: Lightning Web Components Developer Guide: Conditional Rendering, Lightning Web Components Developer Guide: Template Syntax

Question: 95

What are two advantages of using Lightning Data Service?

A. Communicates with other components
B. Converts between different data formats
C. Combines and de-duplicates server calls
D. Loads record data progressively

Answer: C, D

Explanation:

Two advantages of using Lightning Data Service are that it combines and de-duplicates server calls and that it loads record data progressively. Lightning Data Service is a service that provides access to Salesforce data and metadata in Lightning web components. It optimizes performance and minimizes server round trips by caching data on the client side and sharing data across components. It also combines and de-duplicates server calls by batching requests for the same record or object and returning a single response. It also loads record data progressively by returning available cached data first and then fetching updated data from the server asynchronously. Communicating with other components and converting between different data formats are not advantages of using Lightning Data Service, as they are not related to its functionality or features. Salesforce Reference: Lightning Web Components Developer Guide: Lightning Data Service, Lightning Web Components Developer Guide: Work with Salesforce Data

Question: 96

What is likely to happen if a developer leaves debug mode turned on in an environment?

A. The performance of the org will become slower each day
B. The user will begin getting JavaScript limit exceptions
C. The org will turn off debug mode after 72 hours
D. A banner will be displayed to the user indicating that the org is in debug mode

Answer: D

Explanation:

If a developer leaves debug mode turned on in an environment, the user will begin getting JavaScript limit exceptions. Debug mode is a setting that enables more detailed logging and error reporting for Lightning web components. However, it also increases the size and complexity of the JavaScript code that is delivered to the browser, which can cause performance issues and JavaScript limit exceptions. The JavaScript limit exceptions are errors that occur when the browser reaches its maximum capacity for executing JavaScript code, such as memory heap size or script execution time. The performance of the org will not become slower each day, as debug mode only affects the client-side performance, not the server-side performance. The org will not turn off debug mode after 72 hours, as debug mode is a persistent setting that can only be changed manually by an administrator. A banner will not be displayed to the user indicating that the org is in debug mode, as debug mode is a transparent setting that does not affect the user interface. Salesforce Reference: Lightning Web Components Developer Guide: Debug Your Code, Lightning Web Components Developer Guide: JavaScript Limit Exceptions

Question: 97

A developer has created a custom Lightning web component to display on the Product Detail page in the store. When the developer goes to add the component to the page in Experience Builder, it is missing from the list of custom components.

Which XML fragment should the developer include in the component's configuration XML file to ensure the custom component is available to add to the page?

A)

```
<isExposed>true</isExposedTrue>
<targets>
<target>lightningCommunity_Page</target>
<targets>
```

B)

```
<builder>ExperienceCloud</builder>
<target>RecordPage</target>
```

C)

```
<isExposed target='ExperienceCloud >
<pageType>RecordPage</pageType>
</isExposed>
```

D)

```
<isAvailable>true</isAvailable>
<targets>lightningCommunity_RecordPage<targets>
```

A. Option A
B. Option B
C. Option C
D. Option D

Answer: B

Explanation:

The XML fragment that the developer should include in the component's configuration XML file to ensure the custom component is available to add to the page is option B. Option B uses the <targets> tag to specify where the component can be used in an app. The <targets> tag contains a list of <target> tags, each of which defines a valid target for the component. The value of the <target> tag

for the Product Detail page in the store is lightningCommunity RecordPage, which means that the component can be used on any record page in a Lightning community. Option A is incorrect because it uses an invalid value for the <target> tag. There is no such target as lightningCommunity ProductDetailPage. Option C is incorrect because it uses an invalid syntax for the <targets> tag. The <targets> tag should not have any attributes, such as isAvailable. Option D is incorrect because it uses an invalid syntax for the <target> tag. The <target> tag should not be selfclosing, but rather have a closing tag (</target>). Salesforce Reference: Lightning Web Components Developer Guide: Configure Components for Lightning Communities, Lightning Web Components Developer Guide: Configure Components for Different Pages and Apps

Question: 98

Which two statements are accurate about the Cart Item with a Type of Charge?

A. It is created with the Cart Delivery Group Method after the shipping integration
B. It is created with the Cart Delivery Group Method after the freight integration
C. It is linked directly to a Cart Id
D. It is linked directly to a Catalog Id

Answer: CD

Explanation:

Two statements that are accurate about the Cart Item with a Type of Charge are that it is linked directly to a Cart Id and that it is linked directly to a Catalog Id. A Cart Item with a Type of Charge is a special type of Cart Item that represents an additional charge or fee that is applied to a Cart, such as shipping, handling, or tax. A Cart Item with a Type of Charge is linked directly to a Cart Id, which means that it belongs to a specific Cart and can be retrieved or updated along with other Cart Items. A Cart Item with a Type of Charge is also linked directly to a Catalog Id, which means that it references a specific Catalog that contains the products and prices for the store. A Cart Item with a Type of Charge is not created with the Cart Delivery Group Method after the shipping integration or after the freight integration, as these are not related to the creation of Cart Items. The Cart Delivery Group Method is a method that determines how products are grouped into delivery groups based on their shipping methods and addresses. The shipping integration and the freight integration are integrations that calculate and apply shipping costs and freight charges to a Cart or an Order.
Salesforce Reference: B2B Commerce Developer Guide: Cart Item Object, B2B Commerce Developer Guide: Shipping Integration, B2B Commerce Developer Guide: Freight Integration

Question: 99

When a developer configures a tax integration for a store, what happens to the previously calculated tax entries during the checkout flow?

A. Ignored during recalculation
B. Saved prior to recalculation
C. Deleted from the Cart
D. Modified with the new tax calculation

Answer: C

Explanation:

When a developer configures a tax integration for a store, the previously calculated tax entries during the checkout flow are deleted from the Cart. A tax integration is an integration that calculates and applies tax rates and amounts to a Cart or an Order based on various factors, such as product type, price, quantity, location, and tax rules. A tax integration can use either an external tax service provider or custom Apex code to perform the tax calculation. When a developer configures a tax integration for a store, any existing tax entries in the Cart are deleted before calling the tax integration service or method. This ensures that the tax calculation is accurate and up-to-date based on the current state of the Cart and avoids any conflicts or inconsistencies with previous tax entries. The previously calculated tax entries are not ignored during recalculation, saved prior to recalculation, or modified with the new tax calculation, as these are not valid actions for handling existing tax entries. Salesforce Reference: B2B Commerce Developer Guide: Tax Integration, B2B Commerce Developer Guide: Tax Calculation Flow

Question: 100

Which three actions must a developer take, in a B2B Commerce store, to accept credit card payments using a client's chosen payment provider?

A. Create a named credential for authentication with the payment provider.
B. Create a RegisteredExternalService record for the custom payment provider class.
C. Create an Apex class that implements the sfdc_checkout.PaymentGatewayAdapter
D. Create a PaymentProviderGateway record for the custom payment provider class.
E. Create an Apex class that implements the commercepayments.PaymentGatewayAdapter.

Answer: A, C, D

Explanation:

Three actions that a developer must take, in a B2B Commerce store, to accept credit card payments using a client's chosen payment provider are: create a named credential for authentication with the payment provider, create an Apex class that implements the sfdc_checkout.PaymentGatewayAdapter interface, and create a PaymentProviderGateway record for the custom payment provider class. Creating a named credential for authentication with the payment provider allows the developer to securely store and manage authentication information, such as username, password, token, or certificate, for connecting to the payment provider's API or service. Creating an Apex class that implements the sfdc_checkout.PaymentGatewayAdapter interface allows the developer to define custom logic for processing credit card payments using the payment provider's API or service. The interface provides methods for validating credit card information, authorizing payments, capturing payments, voiding payments, and refunding payments. Creating a PaymentProviderGateway record for the custom payment provider class allows the developer to register the custom

payment provider class as a payment gateway in the store and associate it with a payment method. Creating a RegisteredExternalService record for the custom payment provider class is not a required action, as this is only used for registering external services that are not related to payment processing, such as tax or shipping services. Creating an Apex class that implements the commercepayments.PaymentGatewayAdapter interface is not a required action either, as this is only

used for D2C Commerce stores, not B2B Commerce stores. Salesforce Reference: B2B Commerce Developer Guide: Payment Integration, B2B Commerce Developer Guide: Payment Gateway Adapter Interface, B2B Commerce Developer Guide: Payment Provider Gateway Object

Question: 101

What interface must a developer implement to override Tax in Checkout?

A. sfdc.checkout.CartTaxCalculations
B. sfdc.commerce.TaxCalculations
C. sfdc_commerce.CartTaxCalculations
D. sfdc_checkout.TaxCalculations
E. sfdc.commerce.CheckoutTaxCalculations

Answer: D

Explanation:

To override tax in checkout, a developer must implement the sfdc_checkout.TaxCalculations interface. The sfdc_checkout.TaxCalculations interface is an Apex interface that provides methods for customizing the tax calculation logic for a cart or an order. The interface allows the developer to define how to retrieve tax rates and rules, how to apply tax amounts and adjustments, and how to handle tax errors and exceptions. The developer can use the sfdc_checkout.TaxCalculations interface to integrate with a third-party tax service provider or to implement their own tax calculation logic. The other interfaces do not exist or are not related to tax calculation. Salesforce Reference: B2B Commerce Developer Guide: Tax Integration, B2B Commerce Developer Guide: Tax Calculations Interface

Question: 102

What two things happen with the Cart during tax implementation?

A. New entries are written to the Cart
B. Previous entries are copied to another object
C. Previous entries are deleted from the Cart
D. New entries are written to the Order Summary

Answer: A, C

Explanation:

Two things that happen with the cart during tax implementation are that new entries are written to the cart and previous entries are deleted from the cart. A tax implementation is an integration that calculates and applies tax rates and amounts to a cart or an order based on various factors, such as product type, price, quantity, location, and tax rules. A tax implementation can use either an external tax service provider or custom Apex code to perform the tax calculation. When a tax implementation is invoked for a cart, it writes new entries to the cart with a type

of Charge and a charge type of Tax. These entries represent the tax amounts and adjustments that are applied to the cart. Before writing new entries to the cart, the tax implementation deletes any existing entries with a type of Charge

and a charge type of Tax from the cart. This ensures that the tax calculation is accurate and up-to- date based on the current state of the cart and avoids any conflicts or inconsistencies with previous tax entries. Previous entries are not copied to another object or modified with the new tax calculation, as these are not valid actions for handling existing tax entries. Salesforce Reference: B2B Commerce Developer Guide: Tax Integration, B2B Commerce Developer Guide: Tax Calculation Flow

Question: 103

Which cart item type is created with the Cart Delivery Group Method after the shipping integration?

A. Surcharge
B. Charge
C. Shipping
D. Delivery

Answer: B

Explanation:

A cart item with a type of Charge is created with the Cart Delivery Group Method after the shipping integration. A shipping integration is an integration that calculates and applies shipping costs and options to a cart or an order based on various factors, such as product weight, dimensions, quantity, location, and carrier service level. A shipping integration can use either an external shipping service provider or custom Apex code to perform the shipping calculation. The Cart Delivery Group Method is a method that determines how products are grouped into delivery groups based on their shipping methods and addresses. A delivery group is a logical grouping of products that are shipped together using the same shipping method and address. After performing the shipping calculation for each delivery group, the shipping integration creates a cart item with a type of Charge and a charge type of Shipping for each delivery group. These cart items represent the shipping costs and options that are applied to each delivery group. A cart item with a type of Surcharge, Shipping, or Delivery does not exist or is not related to the shipping integration. Salesforce Reference: [B2B Commerce Developer Guide: Shipping Integration], [B2B Commerce Developer Guide: Cart Delivery Group Method]

Question: 104

Ursa Major is planning to implement Salesforce B2B Commerce, and a developer needs to configure taxes for their storefront. The company operates in multiple states, each with different tax rates and tax rules.
What are two ways the developer should configure taxes in B2B Commerce?

A. Configure a tax engine using third-party software.
B. Configure tax rates and rules for each state in Salesforce B2B Commerce.
C. Use a different pricebook for each state.
D. Use the Salesforce out-of-the-box tax calculator.

Answer: A, B

Explanation:

Two ways that a developer should configure taxes in B2B Commerce for Ursa Major are: configure a tax engine using third-party software and configure tax rates and rules for each state in Salesforce B2B Commerce. Configuring a tax engine using third-party software allows the developer to integrate with an external tax service provider that can calculate and apply accurate and up-to-date tax rates and rules for different states and jurisdictions. The developer can use the sfdc_checkout.TaxCalculations interface or the RegisteredExternalService object to connect to the third-party tax service provider's API or service. Configuring tax rates and rules for each state in Salesforce B2B Commerce allows the developer to implement their own custom tax calculation logic using Apex code. The developer can use custom objects, such as TaxRate c and TaxRule c, to store and manage tax rates and rules for different states and products. The developer can also use the sfdc_checkout.TaxCalculations interface to define how to retrieve and apply tax rates and rules for a cart or an order. Using a different pricebook for each state is not a good way to configure taxes in B2B Commerce, as it can create complexity and inconsistency in pricing and discounting logic. Using the Salesforce out-of-the-box tax calculator is not a good way either, as it does not support complex or dynamic tax scenarios that may vary by state or jurisdiction. Salesforce Reference: [B2B Commerce Developer Guide: Tax Integration], [B2B Commerce Developer Guide: Tax Calculations Interface], [B2B Commerce Developer Guide: TaxRate Object], [B2B Commerce Developer Guide: TaxRule Object]

Question: 105

A developer has been working on the flow of an Inventory Class for checkout. The developer has handled all the error states and now needs to indicate that inventory is available for all items and amounts in the cart.
Which step should happen next?

A. Return TRUE
B. Return sfdc_checkout.IntegrationStatus.Status.SUCCESS
C. Return sfdc.checkout.InventoryStatus.Status.SUCCESS
D. Return sfdc.checkout.InventoryStatus.SUCCESS

Answer: C

Explanation:

To indicate that inventory is available for all items and amounts in the cart, the developer should return sfdc.checkout.InventoryStatus.Status.SUCCESS. The sfdc.checkout.InventoryStatus.Status.SUCCESS is a constant value that represents a successful inventory status. It means that the inventory class has checked the availability of all the products in the cart and confirmed that they are in stock and can be fulfilled. The developer should use this value as the return value of the checkInventory method, which is a method that implements the sfdc.checkout.InventoryCheck interface. The checkInventory method is invoked during the checkout flow to validate the inventory availability of the products in the cart. Returning TRUE is not a valid way to indicate inventory availability, as it is not a valid inventory status value. Returning sfdc_checkout.IntegrationStatus.Status.SUCCESS is not a valid way either, as it is not an inventory status value, but rather an integration status value. Returning sfdc.checkout.InventoryStatus.SUCCESS is not a valid way either, as it is not a valid syntax for accessing the inventory status value. Salesforce Reference: B2B Commerce Developer Guide:

Inventory Integration, B2B Commerce Developer Guide: Inventory Status Enum, B2B Commerce Developer Guide: Inventory Check Interface

Question: 106

Universal Containers (UC) is ready to build a tax provider class using the interfaces available in the Buyer Experience SDK. When creating a tax provider, what are three things that a developer should consider first?

A. Steps to complete in the Tax Service
B. How to handle results
C. WhethertouseJSONorXML
D. What to implement
E. What events to fire in the Lightning web component

Answer: A, B, D

Explanation:

When creating a tax provider, three things that a developer should consider first are: steps to complete in the tax service, how to handle results, and what to implement. Steps to complete in the tax service are the actions that the developer needs to perform to connect to and use the third-party tax service provider's API or service. These steps may include authentication, authorization, request formatting, response parsing, error handling, and logging. How to handle results are the actions that the developer needs to perform to process and apply the tax calculation results from the tax service to the cart or order. These actions may include creating or updating cart items with type of Charge and charge type of Tax, applying tax adjustments or exemptions, and displaying tax amounts and details on the storefront. What to implement are the methods that the developer needs to define in their custom Apex class that implements the sfdc_checkout.TaxCalculations interface. The interface provides methods for customizing the tax calculation logic for a cart or an order, such as getTaxRatesAndRules, applyTaxAmountsAndAdjustments, and handleTaxErrorsAndExceptions. Whether to use JSON or XML is not something that the developer should consider first, as it is not a critical or relevant factor for creating a tax provider. It may depend on the format that the tax service provider supports or prefers, but it does not affect the overall functionality or performance of the tax integration. What events to fire in the Lightning web component is not something that the developer should consider first either, as it is not related to creating a tax provider. It may be an optional feature that the developer can add to enhance the user interface or user experience of their storefront, but it does not affect the core logic or functionality of the tax integration. Salesforce Reference: B2B Commerce Developer Guide: Tax Integration, [B2B Commerce Developer Guide: Tax Calculations Interface]

Question: 107

A developer is trying to integrate a new shipping provider to use during checkout in a storefront Which two steps must the developer take to make an integration available for selection?

A. Create a RegisteredExternalService record using Workbench.
B. Create an Apex class that uses the integration framework.
C. Modify the StoreIntegratedService to map to an Apex class ID using Workbench.
D. Enter the integration class name and version in the store administration.

Answer: A, B

Explanation:

To make an integration available for selection, a developer must create a RegisteredExternalService record using Workbench and create an Apex class that uses the integration framework. Creating a RegisteredExternalService record

using Workbench allows the developer to register their custom integration class as an external service in Salesforce B2B Commerce. The RegisteredExternalService record contains information such as the class name, version, display name, description, and category of the integration class. The category determines where and how the integration class can be used in B2B Commerce, such as ShippingService or TaxService. Creating an Apex class that uses the integration framework allows the developer to define custom logic for integrating with an external service provider's API or service. The integration framework provides interfaces and classes for various types of integrations, such as shipping, tax, payment, inventory, and freight. The developer can implement these interfaces and classes in their custom Apex class and override their methods with their own logic. Modifying the StoreIntegratedService to map to an Apex class ID using Workbench is not a required step for making an integration available for selection, as it is only used for registering internal services that are provided by Salesforce B2B Commerce out-of-the-box. Entering the integration class name and version in store administration is not a required step either, as it is only used for selecting an existing integration class that has already been registered as an external service. Salesforce Reference: [B2B Commerce Developer Guide: Integration Framework], [B2B Commerce Developer Guide: RegisteredExternalService Object]

Question: 108

While in the process of gathering requirements from a customer about how they would like to set up their net new storefront checkout experience, a consultant learns that the customer needs the ability to add new shipping and billing addresses during checkout.
Which approach should a developer take to meet this requirement?

A. Create a new shipping address checkout subflow that utilizes the Buyer Managed Contact Point Addresses component.
B. Enable Buyer Managed Contact Point Addresses within the Shipping Address standard component in the Checkout subflow.
C. Enable Buyer Managed Contact Point Addresses within Commerce Administration.
D. Create a Lightning web component that enables this functionality and replaces the current shipping address screen within the Checkout subflow.

Answer: B

Explanation:

To enable the ability to add new shipping and billing addresses during checkout, a developer should enable Buyer Managed Contact Point Addresses within the Shipping Address standard component in the Checkout subflow. The Buyer Managed Contact Point Addresses is a feature that allows customers to add, edit, or delete their shipping and billing addresses during checkout. The developer

can enable this feature by setting the buyerManagedContactPointAddressesEnabled attribute to true in the Shipping Address standard component in the Checkout subflow. The Shipping Address standard component is a component that displays and collects the shipping address information for the cart or order. The Checkout subflow is a subflow that defines the steps and components for the checkout process in the storefront. Creating a new shipping address checkout subflow that utilizes the Buyer Managed Contact Point Addresses component is not a valid way to enable this feature, as there is no such component as Buyer Managed Contact Point Addresses. Enabling Buyer Managed Contact Point Addresses within Commerce Administration is not a valid way either, as this feature is not configurable in Commerce Administration. Creating a Lightning web component that enables this functionality and replaces the current shipping address screen within the Checkout subflow is not a valid way either, as this is unnecessary and complicated when there is already a standard component that supports this feature. Salesforce Reference: [B2B Commerce Developer Guide: Buyer Managed Contact Point Addresses], [B2B Commerce Developer Guide: Shipping Address Component],

[B2B Commerce Developer Guide: Checkout Subflow]

Question: 109

What is the fastest route to setting up a B2B Commerce Store as a developer?

A. Set up B2B Commerce on Lightning Experience manually
B. Create a new store in the Commerce app
C. Import a previously exported store archive
D. Use sfdx setup scripts

Answer: C

Explanation:

The fastest route to setting up a B2B Commerce store as a developer is to use sfdx setup scripts. Sfdx setup scripts are scripts that use Salesforce CLI commands to automate the creation and configuration of a B2B Commerce store. The scripts can perform tasks such as creating scratch orgs, installing packages, importing data, assigning permissions, and deploying code. The scripts can save time and effort for developers who need to set up a B2B Commerce store quickly and easily. Setting up B2B Commerce on Lightning Experience manually is not the fastest route to setting up a B2B Commerce store, as it involves many steps and actions that can be tedious and error-prone. Creating a new store in the Commerce app is not the fastest route either, as it also requires manual configuration and customization of various settings and features. Importing a previously exported store archive is not the fastest route either, as it depends on the availability and quality of the store archive and may not reflect the latest changes or updates. Salesforce Reference: [B2B Commerce Developer Guide: Set Up Your Development Environment], [B2B Commerce Developer Guide: Create Your Store]

Question: 110

Which two behaviors does a target value of lightning FlowScreen in metadata allow for a Lightning web component?

A. It allows the Lightning web component to replace standard functionality in flows and subflows B. It allows the Lightning Web component to be dragged onto a page in Lightning AppBuilder C. It allows the Lightning web component to be used in guided user experiences to gather input D. It automatically generates configuration properties for the Lightning web component

Answer: CD

Explanation:

A target value of lightning FlowScreen in metadata allows for a Lightning web component to be used in guided user experiences to gather input and to automatically generate configuration properties for the Lightning web component. The lightning FlowScreen target value specifies that the component can be used on a flow screen, which is a type of flow element that displays information and collects user input in a flow. A flow is a guided user experience that can automate business processes and guide users through screens, actions, and decisions. By using the lightning FlowScreen target value, the developer can expose their Lightning web component as a custom screen component that can be added to any flow screen. The developer can also use the @api decorator to expose public properties of their Lightning web component as configuration properties that can be set in Flow Builder. Flow Builder is a tool that allows the developer to create and modify flows using a drag-and-drop interface. A target value of lightning FlowScreen does not allow the Lightning web component to replace standard functionality in flows and subflows or to

be dragged onto a page in Lightning App Builder, as these are not related to the flow screen target value. Salesforce Reference: Lightning Web Components Developer Guide: Configure Components for Flows, Lightning Web Components Developer Guide: Communicate with Flow, Lightning Flow Developer Guide: Screen Components Basics

Question: 111

How can a developer introduce new screen behavior in a checkout flow step?

A. Modify the property mappings in checkoutSteps.xml
B. Edit the default subflow directly
C. Clone the appropriate subflow and replace the Lightning web components in it
D. Adjust next-state in previous subflow configuration

Answer: D

Explanation:

To introduce new screen behavior in a checkout flow step, a developer should clone the appropriate subflow and replace the Lightning web components in it. A subflow is a reusable flow that can be invoked from another flow as an element. A checkout flow is a flow that defines the steps and components for the checkout process in the storefront. A checkout flow consists of several subflows, each of which corresponds to a checkout step, such as Cart, Shipping Address, Payment, or Order Summary. To introduce new screen behavior in a checkout flow step, a developer should clone the subflow that matches the checkout step they want to modify and replace the Lightning web components in the cloned subflow with their custom components. The developer can then update the checkoutSteps.xml file to point to the cloned subflow instead of the original subflow. Modifying the property mappings in checkoutSteps.xml is not a valid way to introduce new screen behavior in a

checkout flow step, as it only affects how data is passed between subflows, not how screens are displayed or rendered. Editing the default subflow directly is not a valid way either, as it can cause conflicts or errors with other stores or environments that use the same default subflow. Adjusting next-state in previous subflow configuration is not a valid way either, as it only affects how subflows are transitioned or navigated, not how screens are displayed or rendered. Salesforce Reference: B2B Commerce Developer Guide: Checkout Flow, B2B Commerce Developer Guide: Checkout Subflows, B2B Commerce Developer Guide: Customize Checkout Steps

Question: 112

While working on a commerce rollout, a developer needs to update the checkout process so that buyers can purchase with one of the below payment types.

- Credit Card
- Purchase Order
- Contract Now & Pay Later

Additionally, the developer needs to show only Purchase Order and Contract Now & Pay Later if a custom checkbox field on the account is checked.
How should the developer meet these requirements?

A. Create a custom Lightning web component that can be used with the standard payment component. Use a publish-<g, subscribe (pub-sub) model to listen to events from the standard component to determine which additional payment options should be shown.
B. Create a custom Lightning web component for the checkout flow that has all the options available. Within that

component, pull data from the account to determine which options to show.
C. Modify the standard payment component settings in the checkout screen flow and add the new payment method. Use the component visibility feature in screen flows to fulfill the account-based payment option criteria.
D. Add a new payment gateway through the reference implementation steps so the payment shows up on the checkout payment screen. Configure the different payment options required.

Answer: B

Explanation:

To update the checkout process so that buyers can purchase with one of the below payment types: Credit Card Purchase Order
Contract Now & Pay Later Additionally, show only Purchase Order and Contract Now & Pay Later if a custom checkbox field on the account is checked, a developer should create a custom Lightning web component for the checkout flow that has all the options available. Within that component, pull data from the account to determine which options to show. Creating a custom Lightning web component for the checkout flow allows the developer to define custom logic and user interface for processing payments using different payment types. The developer can use Apex methods or third-party APIs to integrate with payment service providers or payment gateways and handle payment authorization, capture, void, and refund. The developer can also use @wire or @api decorators to get data from the account object and use its properties, such as the custom checkbox field, to determine which payment options to show or hide based on business logic. Creating a custom Lightning web component that can be used with the standard payment component is not a valid way to meet this requirement, as it does not allow the developer to replace or modify the standard payment
component's logic or user interface. Modifying the standard payment component settings in the checkout screen flow and adding the new payment method is not a valid way either, as it does not allow the developer to add custom payment types or conditional logic based on account data. Adding a new payment gateway through the reference implementation steps so the payment shows up on the checkout payment screen is not a valid way either, as it does not allow the developer to add multiple payment options or conditional logic based on account data. Salesforce Reference: B2B Commerce Developer Guide: Payment Integration, B2B Commerce Developer Guide: Payment Component, B2B Commerce Developer Guide: Checkout Subflow

Question: 113

How should data for Lightning web components be provided?

A. A few properties that contain sets (objects) of data
B. One property that contains all data in one set (object)
C. A single property object that contains sets (objects) of data
D. Independent properties that take simpler, primitive values (e.g. String, Number, Boolean, Array)

Answer: D

Explanation:

Data for Lightning web components should be provided as independent properties that take simpler, primitive values (e.g. String, Number, Boolean, Array). Providing data as independent properties allows the developer to expose data as public or private properties of the Lightning web component and communicate data between components or services. Providing data as simpler, primitive values allows the developer to use data types that are supported by JavaScript and Lightning web components and avoid unnecessary or complex conversions or transformations. Providing data as a few properties that contain sets (objects) of data is not a good way to provide data for Lightning web components, as it can

create confusion or inconsistency in data structure and access. Providing data as one property that contains all data in one set (object) is not a good way either, as it can create complexity or inefficiency in data management and manipulation. Providing data as a single property object that contains sets (objects) of data is not a good way either, as it can create redundancy or duplication in data storage and retrieval. Salesforce Reference: Lightning Web Components Developer Guide: Communicate with Properties, Lightning Web Components Developer Guide: Data Types

Question: 114

A developer needs to loop through a series of child components which are tiles. What is the correct syntax for this if the child component is called appTile?

A)

```
template for :each=(apps.data, records) for:item="app >
  <lightning-layout-item key=(app.ld} size="5
flexibility3' auto 1 class3"slds-p-around_xxx-smair>
    <c-app-tile key={app.ld) app-{app)
draggable={tilesAreDraggable} onselected={navigateMaster}>
    </c-app-tlle>
  </lightning-layout-item>
</template>
```

B)

```
<template (for app : apps.data.records)>
  dightning-layout-item key3{app.ld) size3 5
flexibility13' auto" class= slds-p-around_xxx-small >
    <c-app-tile key={app.ld) app={app)
draggable={tilesAreDraggable) onselected={navigateMaster)>
    </c-app-tile>
  </lightning-layout-item>
</template >
```

C)

```
<template for:(app$.data.records}.each() item3 "app">
  <lightning-layout-item key={app.ldj size="5
flexibility^' auto" class- $ld$-p-around_xxx-smaII">
    <c-app-tile key=(app.ld) app={app}
draggable=(tilesAreDraggable) onselected=(navigateMaster}>
    </capp-tile>
  </lightning-layout-item>
</template>
```

D)

```
template for:each=[apps.data.records) item- app >
```

```
<lightning-layout-item key={app.ld) size= 5"
flexibility^' auto" class[3] 'slds-p-aroundjcxx-smaH"*
    <c-aoo-tile kev={aoDJdl aDD={aDD)
```

A. Option A
B. Option B
C. Option C
D. Option D

Answer: A

Explanation:

The correct syntax for looping through a series of child components which are tiles is option A. Option A uses the for:each directive to iterate over a collection of items and render a template block for each item. The for:each directive requires an expression that evaluates to an array or an iterable object and an item alias that represents the current item in the iteration. The item alias can be used to access the item's properties or pass them to child components. In option A, the expression is appTiles, which is an array of objects that represent app tiles, and the item alias is appTile, which represents the current app tile object in the iteration. The appTile object's properties, such as name, description, and icon, are passed to the app-tile child component using attributes. Option B is incorrect because it uses an invalid syntax for the for:each directive. The for:each directive requires a colon (:) after the for keyword, not an equal sign (=). Option C is incorrect because it uses an invalid syntax for the for:each directive. The for:each directive requires an item alias that represents the current item in the iteration, not a key alias that represents the current index in the iteration. Option D is incorrect because it uses an invalid syntax for the template tag. The template tag requires a closing tag (</template>), not a self-closing tag (<template/>). Salesforce Reference: Lightning Web Components Developer Guide: Iterate Over a Collection, Lightning Web Components Developer Guide: Template Syntax

Question: 115

A developer needs to implement specific styling for a standard component on a single page of the B2B Commerce store using an Aura template. The component should use the default style on all other pages
How should the developer implement the required changes over multiple instances?

A. Use a Custom CSS file in a static resource and add the import using the Edit Head Markup Editor in the Experience Builder.
B. Create a Custom Content Layout Lightning web component that imports the custom CSS file. Set up the page to use this Content Layout.
C. Create a Custom Theme Layout Aura component that imports the custom CSS file. Set up the page to use this Theme Layout.
D. Use the Override CSR Editor in the Experience Builder and add the desired CSS to change the styles.

Answer: C

Explanation:

To implement specific styling for a standard component on a single page of the B2B Commerce store using an Aura template, a developer should create a custom theme layout Aura component that imports the custom CSS file and set up the page to use this theme layout. A theme layout is a type of Aura component that defines the header and footer of a page in the storefront. A theme layout can also import custom CSS files from static resources and apply them to the page. A developer can create a custom theme layout Aura component that imports the custom CSS file that contains

the specific styling for the standard component and assign it to the page that needs the custom styling.

This way, the custom styling will only affect the standard component on that page and not on other pages that use a different theme layout. Using a custom CSS file in a static resource and adding the import using the Edit Head Markup Editor in the Experience Builder is not a valid way to implement specific styling for a standard component on a single page, as it will affect all pages that use the same template. Creating a custom content layout Lightning web component that imports the custom CSS file and setting up the page to use this content layout is not a valid way either, as it will not affect the standard component that is outside of the content layout. Using the Override CSR Editor in the Experience Builder and adding the desired CSS to change the styles is not a valid way either, as it will affect all pages that use the same template. Salesforce Reference: B2B Commerce Developer Guide: Theme Layout Component, B2B Commerce Developer Guide: Content Layout Component, B2B Commerce Developer Guide: Override CSR Editor

Question: 116

Which two methods from the platformResourceLoader module are relevant for including third party JavaScript and CSS in a Lightning web component?

A. loadClientScript
B. loadScript
C. loadCss
D. loadStyle

Answer: B, D

Explanation:

Two methods from the platformResourceLoader module that are relevant for including third party JavaScript and CSS in a Lightning web component are loadScript and loadStyle. The platformResourceLoader module is a module that provides methods for loading JavaScript or CSS files from static resources or external URLs into a Lightning web component. The loadScript method is used to load a JavaScript file and execute it in the component. The loadStyle method is used to load a CSS file and apply it to the component. The loadClientScript method does not exist or is not relevant for including third party JavaScript in a Lightning web component. The loadCss method does not exist or is not relevant for including third party CSS in a Lightning web component. Salesforce Reference: Lightning Web Components Developer Guide: Load Scripts and Style Sheets, [Lightning Web Components Developer Guide: platformResourceLoader Module]

Question: 117

In which two ways can events fired from Lightning web components be handled?

A. Programmatically adding event listeners
B. Adding callbacks to components
C. Listening for all possible events at the document root
D. Attaching handlers to DOM elements

Answer: A, D

Explanation:

Two ways that events fired from Lightning web components can be handled are programmatically adding event listeners and attaching handlers to DOM elements. Programmatically adding event listeners is a way of handling events by using JavaScript code to register functions that are invoked when an event occurs. The developer can use methods such as addEventListener or @wire to add event listeners to components or services that fire events. Attaching handlers to DOM elements is a way of handling events by using HTML attributes to bind functions that are invoked when an event occurs. The developer can use attributes such as onclick or onchange to attach handlers to DOM elements that fire events. Adding callbacks to components is not a valid way of handling events fired from Lightning web components, as it is not related to event handling, but rather to asynchronous programming. Listening for all possible events at the document root is not a valid way either, as it is not efficient or recommended for event handling, as it can cause performance issues or conflicts with other event listeners. Salesforce Reference: [Lightning Web Components Developer Guide: Handle Events], [Lightning Web Components Developer Guide: Communicate with Events]

Question: 118

Which two components in a B2B store template should a developer use to customize a storefront page?

A. My Lists
B. Product List
C. Order List
D. Address List

Answer: AB

Explanation:

Two components in a B2B store template that a developer can use to customize a storefront page are My Lists and Product List. My Lists is a standard component that displays and manages lists of products that customers can create, save, or share in the storefront. A developer can use this component to enable customers to organize their favorite or frequently purchased products into lists and access them easily. Product List is a standard component that displays and filters products from one or more product lists in the storefront. A developer can use this component to showcase products from different categories or collections and allow customers to browse or search for products based on various criteria. Order List is not a component in a B2B store template, but rather an object that stores information about orders placed by customers in the storefront. Address List is not a component either, but rather an object that stores information about addresses associated with customers in the storefront. Salesforce Reference: [B2B Commerce Developer Guide: My Lists Component], [B2B Commerce Developer Guide: Product List Component], [B2B Commerce Developer Guide: Order List Object], [B2B Commerce Developer Guide: Address List Object]

Question: 119

A developer has made a component with a lightning combobox in the follow! markup. To handle changes on the combobox, what should replace <CHANGE FVENT>?

```
<template>
  <lightning-combobox
    name= billingAddressSelector" label=(title)
    options={selectableAddresses) value={initialSelectedAddressld} class=
```

```
    slds-p-bottom_medium" onchange=<CHANGE_EVENT> requ»red={bil
    lingAddressRequired} >
    </lightning-combobox>
</template>
```

A. {event:handleChange}
B. javascript:void(0);handleChange();
C. {handleChange()}
D. {handleChange}

Answer: D

Explanation:

To handle changes on the combobox, the developer should replace <CHANGE EVENT> with {handleChange}. The handleChange is a function that is defined in the JavaScript file of the Lightning web component and is invoked when the value of the combobox changes. The developer can use this function to perform custom logic or actions based on the selected value of the combobox, such as updating other components or properties, calling Apex methods or services, or firing events. The developer can use the onchange attribute to bind the handleChange function to the combobox element in the HTML file of the Lightning web component. The onchange attribute is an HTML attribute that specifies a function to be executed when the value of an element changes. {event:handleChange} is not a valid way to handle changes on the combobox, as it is not a valid syntax for binding a function to an element. javascript:void(0);handleChange(); is not a valid way either, as it is not a valid syntax for binding a function to an element. {handleChange()} is not a valid way either, as it is not a valid syntax for binding a function to an element. Salesforce Reference: Lightning Web Components Developer Guide: Handle Events, Lightning Web Components Developer Guide: Communicate with Properties, Lightning Web Components Developer Guide: lightningcombobox

Question: 120

A developer is working in Visual Studio Code on a previously deployed project which is rather large and deployments are time consuming. The developer wants to deploy some small CSS changes without waiting for the entire project deployment. What are two ways this can be accomplished?

A. Right-click the folder for the component and choose Deploy Source to Org
B. Right-click the CSS file that was edited and select Deploy Single File
C. Right-click the CSS file and choose Deploy Source to Org
D. Click the Tools menu and select Deploy styles
E. Deploy the entire project. Only the change will be copied

Answer: A, B

Explanation:

Two ways that a developer can deploy some small CSS changes without waiting for the entire project deployment are right-clicking the folder for the component and choosing Deploy Source to Org and right-clicking the CSS file that was edited and selecting Deploy Single File. Deploying source to org is a way of deploying metadata from a local project to an org using Salesforce CLI commands. The developer can use Visual Studio Code to execute these commands by right-clicking on files or folders in the project and choosing from various deployment options. Right-clicking the folder for the component and choosing Deploy Source to Org allows the developer to deploy only the files that belong to that component, such as HTML, JavaScript, CSS, and XML files. Right-clicking the CSS file that was edited and selecting Deploy

Single File allows the developer to deploy only that CSS file and not any other files in the project. These options can save time and bandwidth for deploying small changes without affecting other components or files in the project. Modifying the StoreIntegratedService to map to an Apex class ID using Workbench is not a way of deploying CSS changes, as it is only used for registering internal services that are provided by Salesforce B2B Commerce out-of-the-box. Entering the integration class name and version in store administration is not a way of deploying CSS changes either, as it is only used for selecting an existing integration class that has already been registered as an external service. Salesforce Reference: Salesforce CLI Command Reference: force:source:deploy, Salesforce Developer Tools for Visual Studio Code, B2B Commerce Developer Guide: Integration Framework, B2B Commerce Developer Guide: RegisteredExternalService Object

Question: 121

Northern Tail Outfitters (NTO) is converting an existing aura component into a Lightning Web Component. The aura component has the following source code:

```
<!— owner.cmp —>
<aura:component>
  <aura attribute name='(Name" type=”String" />
  <c:chlld fir$tName="(!v.fName)7>
</aura:component>
```

What is the equivalent of this code in a Lightning Web Component?

A)

```
<!~ owner.html ~>
<template>
  <c-child fir$t-name=(!fName}x/child>
</template>
```

B)

```
<!- owner.html —>
<template>
  <c<hild first-name=(fName}x/child>
</template>
```

C)

```
<!~ owner.html ->
<template>
  <c-child first-name=(<fName>}x/chlld> </template>
```

D)

```
<!- owner.html ->
<template>
  <c-child first-name*{item.fName}x/child> </template>
```

A. Option A

B. Option B

C. Option C

D. Option D

Answer: B

Explanation:

The equivalent of this code in a Lightning web component is option B. Option B uses the @api decorator to expose firstName as a public property of the Lightning web component and communicate it with other components or services. The @api decorator is a decorator that marks a property or method as public, which means that it can be accessed by other components or services that use or consume this component. The @api decorator also makes the property reactive, which means that it can track changes and update the component accordingly. In option B, firstName is exposed as a public property using the @api decorator and passed to the child element using an attribute. Option A is incorrect because it uses an invalid syntax for exposing firstName as a public property. The @api decorator should be used before the property declaration, not after it. Option C
is incorrect because it uses an invalid syntax for exposing firstName as a public property. The @api decorator should be used with parentheses, not without them. Option D is incorrect because it uses an invalid syntax for exposing firstName as a public property. The @api decorator should be used with camel case, not with hyphens. Salesforce Reference: Lightning Web Components Developer Guide: Communicate with Properties, Lightning Web Components Developer Guide: Decorators

Question: 122

An administrator has just provided a developer with a completely new org and a username.
Assuming the username is me@my-new-org.org, what is the correct set of steps to authorize the org for Command Line Interface (CLI) access so the developer can deploy Lightning web components?

A. Run the command: 'sfdx force:login -r "https://login.salesforce.com'' and supply the credentials in the browser when it opens.

B. Run the command 'sfdx force:auth:web:login -a "https://login.salesforce.com'"' and then supply the credentials in the browser when it opens.

C. Run the command: 'sfdx force:auth:web:login -r "https://login.salesforce.com" and then supply the credentials in the ^ browser when it opens ^

D. Run the command 'sfdx force:auth:web:login -r "https://login.salesforce.com" -username^'mefaJmy-new-org.org"'

Answer: C

Explanation:

To authorize the org for Command Line Interface (CLI) access so the developer can deploy Lightning web components, the developer should run the command: 'sfdx force:auth:web:login -r “https://login.salesforce.com” and then supply the credentials in the browser when it opens. The sfdx force:auth:web:login command is a Salesforce CLI command that authorizes an org using the web server flow. The web server flow is an OAuth 2.0 authentication flow that opens a browser window and prompts the user to log in to Salesforce and allow access to the CLI. The -r flag specifies the login URL of the org, which is https://login.salesforce.com for production or developer orgs. Running this command will open a browser window and ask the developer to enter their username and password for the org. After successfully logging

in, the developer will be able to use the CLI to perform various tasks with the org, such as deploying or retrieving metadata, running tests, or executing commands. Running the command: 'sfdx force:login -r "https://login.salesforce.com" is not a valid way to authorize the org for CLI access, as there is no such command as sfdx force:login. Running the command 'sfdx force:auth:web:login -a "https://login.salesforce.com"' is not a valid way either, as the -a flag specifies an alias for the org, not a login URL. Running the command 'sfdx force:auth:web:login -r "https://login.salesforce.com" -username'me@my-new-org.org"' is not a valid way either, as there is no such flag as -username. Salesforce Reference: Salesforce CLI Command Reference: force:auth:web:login, Salesforce Developer Tools for Visual Studio Code

Question: 123

A developer is on a tight timeline and needs to implement a Lightning web component which can read, create and modify single records. What is the recommended path forward?

A. Use base components
B. Write custom functions against a wire adapter
C. Create an Apex Controller
D. Use Lightning Data Service

Answer: D

Explanation:

To implement a Lightning web component which can read, create and modify single records, the recommended path forward is to use Lightning Data Service. Lightning Data Service is a service that provides access to Salesforce data and metadata, cache management, and data synchronization across components. Lightning Data Service allows developers to use standard components or base components to perform CRUD (create, read, update, delete) operations on single records without writing Apex code or SOQL queries. Lightning Data Service also handles data caching, performance optimization, and conflict resolution automatically. Using base components is not a sufficient way to implement a Lightning web component which can read, create and modify single records, as base components are only user interface elements that do not provide data access or manipulation functionality by themselves. Base components need to be used with Lightning Data Service or other services to perform CRUD operations on single records. Writing custom functions against a wire adapter is not a recommended way to implement a Lightning web component which can read, create and modify single records, as it involves writing complex and error-prone JavaScript code that may not be efficient or scalable. Writing custom functions against a wire adapter also requires handling data caching, performance optimization, and conflict resolution manually. Creating an Apex controller is not a recommended way either, as it involves writing Apex code that may not be necessary or optimal for performing CRUD operations on single records. Creating an Apex controller also requires exposing Apex methods using @AuraEnabled or @RemoteAction annotations and invoking them from JavaScript code using imperative calls or promises. Salesforce Reference: Lightning Web Components Developer Guide: Access Salesforce Data, Lightning Web Components Developer Guide: Base Components, [Lightning Web Components Developer Guide: Call Apex Methods]

Question: 124

A developer is working in Visual Studio Code on a previously deployed project which is rather large and deployments are time consuming. The developer wants to know if a CSS file containing small changes was actually deployed to the org. What is one way this can be accomplished?

A. Right-click the CSS file and choose Diff File Against Org

B. Click the Tools menu and select Diff Styles Against Org...
C. Right-click the folder for the component and choose Diff Styles Against Org
D. Right-click the folder for the component and choose Diff Files Against Org

Answer: A

Explanation:

To know if a CSS file containing small changes was actually deployed to the org, one way that a developer can accomplish this is by right-clicking the CSS file and choosing Diff File Against Org. Diff

File Against Org is an option that allows the developer to compare a local file with its remote version in the org using Salesforce CLI commands. The developer can use Visual Studio Code to execute these commands by right-clicking on files or folders in the project and choosing from various diff options. Right-clicking the CSS file and choosing Diff File Against Org allows the developer to see the differences between the local CSS file and the remote CSS file in the org side by side in Visual Studio Code. This way, the developer can verify if their changes were deployed successfully or not. Clicking the Tools menu and selecting Diff Styles Against Org… is not a valid way to know if a CSS file was deployed to the org, as there is no such option in Visual Studio Code or Salesforce CLI. Right-clicking the folder for the component and choosing Diff Styles Against Org is not a valid way either, as there is no such option in Visual Studio Code or Salesforce CLI. Right-clicking the folder for the component and choosing Diff Files Against Org is not a valid way either, as it will compare all the files in the folder, not just the CSS file, which may not be efficient or necessary. Salesforce Reference: [Salesforce CLI Command Reference: force:source:diff], [Salesforce Developer Tools for Visual Studio Code]

Question: 125

A product is assigned to the entitlement policy but is missing from the Price Book related to the Buyer Group. The entitlement policy has View products and View prices in catalog checked.
How will the product behave on the B2B Portal?

A. The product will not appear on the Portal but can be searched since it is part of the entitlement policy.
B. The product will appear on the Portal with Price Unavailable status and can also be added to the cart.
C. The product will not appear on the Portal and therefore cannot be added to the cart.
D. The product will appear on the Portal with Price Unavailable status but cannot be added to the cart.

Answer: C

Explanation:

The product will not appear on the Portal and therefore cannot be added to the cart. A product is assigned to the entitlement policy but is missing from the Price Book related to the Buyer Group. The entitlement policy has View products and View prices in catalog checked. An entitlement policy is a set of rules that determines which products and prices a buyer group can access in the storefront. A buyer group is a group of buyers that share the same business relationship with the seller. A price book is a collection of prices for products that are available for purchase in the storefront. A product is a record that represents an item that can be sold in the storefront. To make a product visible and purchasable in the storefront, it must be assigned to both an entitlement policy and a price book that are related to the buyer group. If a product is assigned to an entitlement policy but not to a price book, it will not appear on the Portal and therefore cannot be added to the cart, regardless of the entitlement policy settings. The product will appear on the Portal with Price Unavailable status and can also be added to the cart is not a correct answer, as it contradicts the behavior of a product that is not in a price book. The product will appear on the Portal with Price Unavailable status but

cannot be added to the cart is not a correct answer either, as it also contradicts the behavior of a product that is not in a price book. The product will not appear on the Portal but can be searched since it is part of the entitlement policy is not a correct answer either, as it also contradicts the behavior of a product that is not in a price book. Salesforce Reference: [B2B Commerce Developer Guide:

Entitlement Policy Object], [B2B Commerce Developer Guide: Buyer Group Object], [B2B Commerce Developer Guide: Price Book Object], [B2B Commerce Developer Guide: Product Object]

Question: 126

A developer exports data from an org on a standard entity which has a custom attribute. When they launch Data Loader, select the entity, click the Select All Fields button and click Finish, the custom field they added called MyCustomField_c has no values and no column header in the CSV file. What is the root cause?

A. The user needs to install a specific Zulu JDK that is recommended by Salesforce.
B. A mapping file was not used when the data was loaded in
C. The user does not have access to the field
D. The user has rights to the field but there are no values in it

Answer: C

Explanation:

The root cause of why the custom field MyCustomField c has no values and no column header in the CSV file is that the user does not have access to the field. A user's access to a field is determined by their profile and permission sets, which define their field-level security settings. Field-level security settings control whether a user can see, edit, or delete the value for a particular field on an object. If a user does not have access to a field, they will not be able to view or modify its value in any interface, including Data Loader. Data Loader is a tool that allows users to import or export data between Salesforce and CSV files. When using Data Loader to export data from an org, Data Loader will only include fields that are accessible to the user based on their field-level security settings. If a user does not have access to a field, Data Loader will not include that field in the CSV file, neither as a column header nor as a value. The user needs to install a specific Zulu JDK that is recommended by Salesforce is not the root cause of why MyCustomField c has no values and no column header in the CSV file, as it is not related to field access or Data Loader functionality. A mapping file was not used when the data was loaded in is not the root cause either, as it is not related to field access or Data Loader functionality. A mapping file is an optional file that maps fields between Salesforce objects and CSV files when using Data Loader to import or export data. The user has rights to the field but there are no values in it is not the root cause either, as it contradicts the fact that MyCustomField c has no column header in the CSV file. If the user had rights to the field but there were no values in it, Data Loader would still include MyCustomField c as a column header in the CSV file, but leave its values blank. Salesforce Reference: [Data Loader Guide: Export Data from Salesforce], [Data Loader Guide: Field Mapping], [Salesforce Help: Set Field-Level Security]

Question: 127

A dev at Northern Trail Outfitters (NTO) exported Order Summary records via Data Loader, but noticed that some orders were missing. What is the most likely cause?

A. The export job did not fully complete

B. Order Life Cycle Type was Managed
C. The user does not have rights to some of the records

D. The Status was still set to Draft

Answer: C

Explanation:

The most likely cause of why some orders were missing from the Data Loader export is that the Status was still set to Draft. The Status is a field on the Order Summary object that indicates the current state of the order. The Status can have values such as Draft, Submitted, Confirmed, or Cancelled. A Draft order is an order that has not been submitted or confirmed by the customer or the seller. A Draft order is not considered a completed or valid order and is not included in reports or exports. When using Data Loader to export data from an org, Data Loader will only include orders that have a Status other than Draft, such as Submitted or Confirmed. If an order has a Status of Draft, Data Loader will not include it in the CSV file, which may result in missing orders. The export job did not fully complete is not a likely cause of why some orders were missing from the Data Loader export, as it is not related to the order status or data filtering. Order Life Cycle Type was Managed is not a likely cause either, as it is not related to the order status or data filtering. The Order Life Cycle Type is a field on the Order Summary object that indicates whether the order is managed by Salesforce Order Management or by an external system. The user does not have rights to some of the records is not a likely cause either, as it contradicts the fact that some orders were exported successfully. If the user did not have rights to some of the records, Data Loader would not be able to access or export any orders at all. Salesforce Reference: B2B Commerce Developer Guide: Order Summary Object, B2B Commerce Developer Guide: Order Status Enum, Data Loader Guide: Export Data from Salesforce

Question: 128

A developer attempts to export data from an org by launching Data Loader, selecting a standard entity, clicking the "Select All Fields" button and clicking the Finish button. The developer finds that the CustomField_c field they added to the entity has no values under the header in the CSV file output. What is the root cause?

A. The developer does not have the correct JDK that is recommended by Salesforce and this is known to cause issues with exporting custom attributes
B. The developer does not have access to the object's metadata
C. The field is not populated
D. The user does not have rights to the custom field

Answer: D

Explanation:

The root cause of why the CustomField c field they added to the entity has no values under the header in the CSV file output is that the user does not have rights to the custom field. A user's access to a field is determined by their profile and permission sets, which define their field-level security settings. Field-level security settings control whether a user can see, edit, or delete the value for a particular field on an object. If a user does not have access to a field, they will not be able to view or modify its value in any interface, including Data Loader. Data Loader is a tool that allows users to import or export data between Salesforce and CSV files. When using Data Loader to export data from

an org, Data Loader will only include fields that are accessible to the user based on their field-level security settings. If a

user does not have access to a field, Data Loader will not include that field in the CSV file, neither as a column header nor as a value. The developer does not have the correct JDK that is recommended by Salesforce and this is known to cause issues with exporting custom attributes is not the root cause of why CustomField c has no values under the header in the CSV file output, as it is not related to field access or Data Loader functionality. The developer does not have access to the object's metadata is not the root cause either, as it is not related to field access or Data Loader functionality. The object's metadata defines its structure and properties, such as fields, relationships, and layouts. The field is not populated is not the root cause either, as it contradicts the fact that CustomField c has no column header in the CSV file output. If the field was not populated but accessible to the user, Data Loader would still include CustomField c as a column header in the CSV file, but leave its values blank. Salesforce Reference: Data Loader Guide: Export Data from Salesforce, [Data Loader Guide: Field Mapping], [Salesforce Help: Set Field-Level Security]

Question: 129

What is true about mapping custom fields from Cart to Order Summary?

A. A custom field must exist in the Cart and Order Summary objects only to be mapped successfully.
B. The automatic Cart to Order mapping of custom fields can be disabled.
C. All data types are supported for custom fields to be mapped from Cart to Order.
D. There is a limit of 25 custom fields on a Cart that can be mapped to Order.

Answer: A

Explanation:

The correct answer for what is true about mapping custom fields from Cart to Order Summary is that the automatic Cart to Order mapping of custom fields can be disabled. A custom field is a field that is added by a developer or an administrator to an object to store additional information or data. A Cart is an object that represents a collection of products and charges that a customer intends to purchase in the storefront. An Order Summary is an object that represents a confirmed purchase of products and charges by a customer in the storefront. A Cart can be converted to an Order Summary when the customer completes the checkout process and confirms their order. By default, Salesforce B2B Commerce automatically maps custom fields from Cart to Order Summary when converting a Cart to an Order Summary. This means that any custom fields that exist on both Cart and Order Summary objects with identical API names and data types will have their values copied from Cart to Order Summary during the conversion. The automatic Cart to Order mapping of custom fields can be disabled by setting the B2BCommerce.CartToOrderMappingEnabled custom setting to false. This will prevent any custom fields from being copied from Cart to Order Summary during the conversion. A custom field must exist in the Cart and Order Summary objects only to be mapped successfully is not true, as it is not the only requirement for mapping custom fields from Cart to Order Summary. The custom fields must also have identical API names and data types, and the automatic Cart to Order mapping of custom fields must be enabled. All data types are supported for custom fields to be mapped from Cart to Order is not true, as some data types are not supported for mapping custom fields from Cart to Order Summary. The supported data types are Boolean, Date, DateTime, Double, Integer, Long, Percent, String, and TextArea. There is a limit of 25 custom fields on a Cart that can be mapped to Order is not true, as there is no such limit for mapping custom fields from Cart to Order Summary. Any number of custom fields that meet the mapping requirements can be mapped from

Cart to Order Summary. Salesforce Reference: [B2B Commerce Developer Guide: Custom Field Mapping], [B2B Commerce Developer Guide: Cart Object], [B2B Commerce Developer Guide: Order Summary Object]

Question: 130

A developer needs to import some new product data contained in a JSON file one time. What are two viable ways to do this? .

A. Convert the JSON to an xlsx file and use Workbench to import it
B. Run a command like: sfdx force:data:tree:import -f NewProducts.json -u <your username>
C. Convert the JSON to a CSV file and use Data Loader to import it
D. Run a command like: sfdx force:data;import:bulk -f NewProducts.json -u <your username>

Answer: B, C

Explanation:

Two viable ways that a developer can import some new product data contained in a JSON file one time are running a command like: sfdx force:data:tree:import -f NewProducts.json -u <your username> and converting the JSON to a CSV file and using Data Loader to import it. Running a command like: sfdx force:data:tree:import -f NewProducts.json -u <your username> allows the developer to import data from a JSON file into an org using Salesforce CLI commands. The sfdx force:data:tree:import command is a Salesforce CLI command that imports data into an org using JSON files that conform to the SObject Tree API specification. The SObject Tree API specification is a format that defines how records are represented in JSON files for data import or export. The -f flag specifies the path of the JSON file that contains the data to be imported. The -u flag specifies the username or alias of the org where the data will be imported. Running this command will create records in the org based on the data in the JSON file. Converting the JSON to a CSV file and using Data Loader to import it allows the developer to import data from a CSV file into an org using Data Loader. Data Loader is a tool that allows users to import or export data between Salesforce and CSV files. The developer can use an online converter or a spreadsheet application to convert their JSON file into a CSV file that matches the structure and format of their Salesforce object. The developer can then use Data Loader to import the CSV file into their org and create records based on the data in the CSV file. Converting the JSON to an xlsx file and using Workbench to import it is not a viable way to import some new product data contained in a JSON file one time, as Workbench does not support xlsx files for data import or export. Workbench is a web-based tool that provides access to various Salesforce features and functionalities, such as data manipulation, REST Explorer, and Apex Execute. Running a command like: sfdx force:data;import:bulk -f NewProducts.json -u <your username> is not a viable way either, as there is no such command as sfdx force:data;import:bulk. The correct command for importing data using bulk API is sfdx force:data:bulk:upsert. Salesforce Reference: [Salesforce CLI Command Reference: force:data:tree:import], [Salesforce Developer Tools for Visual Studio Code], [Data Loader Guide: Import Data into Salesforce], [Workbench], [Salesforce CLI Command Reference: force:data:bulk:upsert]

Question: 131

A developer has the task to bring some historical data into an org. The data must reside in the org,

but cannot be populated in standard or custom objects. The customer is fine with developers building Ul components to surface this data on a case-by-case basis.
Which option allows a developer to meet all of these requirements?

A. Big objects
B. Lightning Canvas
C. External objects

D. Lightning Out

Answer: C

Explanation:

To bring some historical data into an org, the data must reside in the org, but cannot be populated in standard or custom objects, and the customer is fine with developers building UI components to surface this data on a case-by-case basis, the option that allows a developer to meet all of these requirements is big objects. Big objects are a type of object that can store and manage massive amounts of data on the Salesforce platform. Big objects can store up to billions of records and are accessible through a subset of SOQL or custom user interfaces. Big objects are not subject to the same storage limits or performance issues as standard or custom objects and are suitable for storing historical or archived data that does not need to be updated frequently. Big objects can be defined using Metadata API or declaratively in Setup. Lightning Canvas is not an option that allows a developer to meet all of these requirements, as it is a framework that allows developers to integrate third-party applications into Salesforce. Lightning Canvas does not store data in the org, but rather displays data from external sources using an iframe. External objects are not an option either, as they are a type of object that map to data stored outside Salesforce. External objects do not store data in the org, but rather access data from external systems using OData services. Lightning Out is not an option either, as it is a feature that allows developers to use Lightning components outside Salesforce. Lightning Out does not store data in the org, but rather renders components on external web pages or applications. Salesforce Reference: Salesforce Help: Define Big Objects, Salesforce Help: Lightning Canvas Developer's Guide, Salesforce Help: External Objects, Salesforce Developer Blog: Lightning Out

Question: 132

Which three data types are supported for custom fields while using CSV file format for importing data for a store?

A. Text Area(Long)
B. Picklist (Multi-Select)
C. Lookup Relationship
D. Address
E. Currency

Answer: A, C, E

Explanation:

Three data types that are supported for custom fields while using CSV file format for importing data

for a store are Text Area(Long), Lookup Relationship, and Currency. A custom field is a field that is added by a developer or an administrator to an object to store additional information or data. A data type is a property that defines the type, format, and validation rules of a field. A CSV file is a file format that stores tabular data in plain text using commas to separate values. A store is a record that represents a B2B or B2C storefront in Salesforce. Text Area(Long) is a data type that allows users to enter up to 131,072 characters on separate lines. Text Area(Long) is supported for custom fields while using CSV file format for importing data for a store. Lookup Relationship is a data type that allows users to create a relationship between two objects and select a value from another record. Lookup Relationship is supported for custom fields while using CSV file format for importing data for a store. Currency is a data type that allows users to enter currency values and automatically convert them based on the user's locale and currency settings. Currency is supported for custom fields while using CSV file format for importing data for a store. Picklist (Multi-Select) is a data type that allows users to select one or more values from a predefined list of values. Picklist

(Multi-Select) is not supported for custom fields while using CSV file format for importing data for a store. Address is a data type that allows users to enter address values and automatically format them based on the user's locale settings. Address is not supported for custom fields while using CSV file format for importing data for a store. Salesforce Reference: [Salesforce Help: Custom Field Attributes], [Salesforce Help: Data Types], [Data Loader Guide: Import Data into Salesforce], [B2B Commerce Developer Guide: Store Object]

Question: 133

Northern Trail Outfitters (NTO) has acquired a company and is looking to manage product data across the org seamlessly. The company has a governance policy to not install any tool or use third-party API applications to export or import the data into Salesforce. However, users have access to Salesforce CLI.

Which set of tasks must a developer perform whento export data from Salesforce or import data into Salesforce?

A. sfdx force:data:bulk:export -Product2 -all 0 and
sfdx force:data:bulk:lmport -f Product2.json -all
B. sfdx force:data;tree:export -Product2 -all q and
sfdx force:data:tree:lmport -f Product2.json -all
C. sfdx force:tree:data:export -q "SELECT Id, Name FROM Product2" -u <your username> ® and sfdx force:tree:data:import -f Product2Json -all
D. sfdx force:data:tree:export -q "SELECT Id, Name FROM Product2" -u "<your username>" and sfdx force:data:tree:import -f Product2.json -u w<your username>"

Answer: D

Explanation:

The correct answer for how to export data from Salesforce or import data into Salesforce using Salesforce CLI commands is running a command like: sfdx force:data:tree:export -q "SELECT Id, Name FROM Product2" -u "<your username>" and sfdx force:data:tree:import -f Product2.json -u "<your username>". The sfdx force:data:tree:export command is a Salesforce CLI command that exports data from an org into JSON files that conform to the SObject Tree API specification. The SObject Tree API specification is a format that defines how records are represented in JSON files for data import or export. The -q flag specifies the SOQL query that selects the records and fields to be exported. The -u

flag specifies the username or alias of the org where the data will be exported from. Running this command will generate JSON files that contain the data from the org based on the SOQL query. The sfdx force:data:tree:import command is a Salesforce CLI command that imports data into an org using JSON files that conform to the SObject Tree API specification. The -f flag specifies the path of the JSON file that contains the data to be imported. The -u flag specifies the username or alias of the org where the data will be imported to. Running this command will create records in the org based on the data in the JSON file. Running a command like: sfdx force:data:bulk:export -Product2 -all 0 and sfdx force:data:bulk:import -f Product2.json -all is not a correct answer, as it uses invalid syntax and flags for the sfdx force:data:bulk:export and sfdx force:data:bulk:import commands. The correct syntax and flags for these commands are sfdx force:data:bulk:upsert -s Product2 -f Product2.csv -w 10 -u <your username> and sfdx force:data:bulk:status -i <job ID> -u <your username>. Running a command like: sfdx force:data;tree:export -Product2 -all q and sfdx force:data:tree:import -f Product2.json -all is not a correct answer either, as it uses invalid syntax and flags for the sfdx force:data:tree:export and sfdx force:data:tree:import commands. The correct syntax and flags for these commands are sfdx force:data:tree:export -q "SELECT Id, Name FROM Product2" -u <your username> and sfdx force:data:tree:import -f Product2.json -u <your username>. Running a command like: sfdx force:tree:data:export -q "SELECT Id, Name FROM Product2" -u <your username> ® and sfdx force:tree:data:import -f Product2Json -all is not a correct answer either, as there is no such command as sfdx force:tree:data:export or sfdx force:tree:data:import. The correct commands are sfdx

force:data:tree:export and sfdx force:data:tree:import. Salesforce Reference: [Salesforce CLI Command Reference: force:data:tree:export], [Salesforce CLI Command Reference: force:data:tree:import], [Salesforce CLI Command Reference: force:data:bulk], [Salesforce Developer Tools for Visual Studio Code]

Question: 134

A developer is working on a storefront and is seeing unexpected UI behavior in one of the custom Lightning web components (LWCs) their team has built.
How should the developer investigate the issue?

A. Enable Debug Mode for a storefront user, log in to the storefront, and use Browser Inspection tools and debugger points.
B. Enable Debug Mode for a storefront user, load the LWC in Visual Studio (VS) Code, attach to session, and view debug logs in VS Code.
C. Enable debug logs for a storefront user, log in to storefront and perform action, and view debug logs in Setup.
D. Identify the user, inputs, and failure, then ask Salesforce support to investigate the issue with the custom LWC.

Answer: A

Explanation:

To investigate the issue of seeing unexpected UI behavior in one of the custom Lightning web components (LWCs) their team has built, the developer should enable Debug Mode for a storefront user, log in to the storefront, and use Browser Inspection tools and debugger points. Debug Mode is a feature that allows developers to debug and troubleshoot custom LWCs in the storefront by disabling performance optimizations and enabling source maps. Source maps are files that map the minified or obfuscated code to the original source code, making it easier to read and debug. To

enable Debug Mode for a storefront user, the developer can go to Setup, enter Users in the Quick Find box, select Users, click Edit next to the user name, and select Debug Mode. After enabling Debug Mode, the developer can log in to the storefront as the user and use Browser Inspection tools and debugger points to inspect and debug the custom LWC. Browser Inspection tools are tools that are built into web browsers that allow developers to examine and modify the HTML, CSS, JavaScript, and other aspects of a web page. Debugger points are statements that are added to the JavaScript code of a LWC that pause the execution of the code at a certain point and allow the developer to inspect the values of variables, expressions, and other elements. Enable Debug Mode for a storefront user, load the LWC in Visual Studio (VS) Code, attach to session, and view debug logs in VS Code is not a valid way to investigate the issue of seeing unexpected UI behavior in one of the custom LWCs their team has built, as it is not possible to attach to a session or view debug logs for LWCs in VS Code. Enable debug logs for a storefront user, log in to storefront and perform action, and view debug logs in Setup is not a valid way either, as debug logs do not capture information about LWCs or UI behavior. Debug logs are records of database operations, system processes, and errors that occur when executing a transaction or running unit tests. Identify the user, inputs, and failure, then ask Salesforce support to investigate the issue with the custom LWC is not a valid way either, as it is not a recommended or efficient way of debugging or troubleshooting custom LWCs. Salesforce support may not be able to provide assistance or guidance for custom LWCs that are developed by third-party developers. Salesforce Reference: B2B Commerce Developer Guide: Debug Lightning Web Components, Lightning Web Components Developer Guide: Debug Your Code, Salesforce Help: Debug Logs

Question: 135

During checkout flow customizations, a developer receives an error on shipping cost calculation integrations with the error code: INSUFFICIENT_ACCESS_OR_READONLY.
What is causing this error?

A. The storefront user does not have access to the Cart Delivery Method object.
B. An error has occurred during the cart shipping charge integration.
C. The storefront user does not have access to custom fields on the Order Delivery Method object. D. The cart is no longer in a valid Checkout State.

Answer: D

Explanation:

The error code INSUFFICIENT_ACCESS_OR_READONLY is caused by the cart being no longer in a valid Checkout State during checkout flow customizations. A cart is an object that represents a collection of products and charges that a customer intends to purchase in the storefront. A cart has a Checkout State field that indicates the current state of the checkout process for the cart. The Checkout State can have values such as Draft, InProgress, Completed, or Cancelled. A cart can only be modified or updated when it is in Draft or InProgress state. A cart cannot be modified or updated when it is in Completed or Cancelled state. If an attempt is made to modify or update a cart that is in Completed or Cancelled state, an error with the code INSUFFICIENT_ACCESS_OR_READONLY will be thrown. This error means that the user does not have permission to edit or delete a record because it is read-only or locked. The storefront user does not have access to the Cart Delivery Method object is not a cause of this error code, as it is not related to the cart checkout state or data modification. The Cart Delivery Method object is an object that stores information about the delivery method selected for a

cart in the storefront. An error has occurred during the cart shipping charge integration is not a cause of this error code either, as it is not related to the cart checkout state or data modification. The cart shipping charge integration is an integration that calculates and applies shipping charges to a cart based on various factors such as delivery method, location, weight, volume, etc. The storefront user does not have access to custom fields on the Order Delivery Method object is not a cause of this error code either, as it is not related to the cart checkout state or data modification. The Order Delivery Method object is an object that stores information about the delivery method selected for an order summary in the storefront. Salesforce Reference: B2B Commerce Developer Guide: Cart Object, [B2B Commerce Developer Guide: Cart Delivery Method Object], [B2B Commerce Developer Guide: Order Delivery Method Object], [Salesforce Help: Common Error Messages]

Question: 136

Based on error emails flowing in, a developer suspects that recent edits made to a checkout flow have created a defect. The developer has data points available to use as inputs in reproducing the scenario.
What should the developer do next?

A. Open the flow, select Debug, provide the session ID for replay, and select Run.
B. Open the flow, select Attach to Live Session, provide the session ID, and select Attach.
C. Open the flow, select Debug, provide the Input values, and select Run.
D. Open the flow, select Debug with Inputs, provide the Input values, and select Run.

Answer: A

Explanation:

The next step that the developer should do after suspecting that recent edits made to a checkout flow have created a defect and having data points available to use as inputs in reproducing the scenario is to open the flow, select Debug, provide the Input values, and select Run. A flow is a type of application that automates a business process by collecting data and performing actions in Salesforce or an external system. A flow can be used to customize the checkout process in the storefront by defining the steps and logic that are executed when a customer places an order. A flow can be edited or modified using Flow Builder, a point-and-click tool that allows developers to create and manage flows. Flow Builder also provides debugging and testing tools that allow developers to run and troubleshoot flows before deploying them. To debug or test a flow, the developer can open the flow in Flow Builder, select Debug from the toolbar, provide the Input values for the flow variables, and select Run. This will execute the flow in debug mode, which simulates how the flow runs in the org with real data. The developer can use debug mode to verify if the flow works as expected or if there are any errors or issues with the flow logic or actions. Open the flow, select Attach to Live Session, provide the session ID, and select Attach is not a valid next step, as it is not a feature or option available in Flow Builder or Salesforce CLI. Attach to Live Session is a feature that allows developers to attach a debugger to a running Apex session and inspect the state of the code execution. Open the flow, select Debug with Inputs, provide the Input values, and select Run is not a valid next step either, as it is not a feature or option available in Flow Builder or Salesforce CLI. Debug with Inputs is a feature that allows developers to debug an Apex class or trigger with predefined input values and breakpoints. Open the flow, select Debug, provide the session ID for replay, and select Run is not a valid next step either, as it is not a feature or option available in Flow Builder or Salesforce CLI. Replay is a feature that allows developers to replay an Apex log file and inspect the

state of the code execution at each line. Salesforce Reference: [B2B Commerce Developer Guide: Customize Checkout Flows], [Salesforce Help: Flow Builder], [Salesforce Help: Debug Your Flows], [Salesforce Developer Blog: Apex Replay Debugger]

Question: 137

Which two technologies can subscribe to the CommerceDiagnosticEvents event?

A. Aura Components
B. Processes
C. Streaming API
D. Lightning web components

Answer: A, D

Explanation:

Two technologies that can subscribe to the CommerceDiagnosticEvents event are Aura Components and Lightning web components. CommerceDiagnosticEvents is an event that is fired by Salesforce B2B Commerce when an error occurs in the storefront. CommerceDiagnosticEvents contains information about the error, such as error code, error message, error type, and error details. CommerceDiagnosticEvents can be subscribed by other components or services that want to handle or display the error information in different ways. Aura Components are a type of component that can be used to create custom user interfaces for Salesforce apps. Aura Components can subscribe to CommerceDiagnosticEvents using an aura:handler tag in their markup file. The aura:handler tag specifies an event name, an action attribute that defines a controller function to handle the event, and other optional attributes. Lightning web components are another type of component that can be used to create custom user interfaces for Salesforce apps.

Lightning web components can subscribe to CommerceDiagnosticEvents using an @wire decorator in their JavaScript file. The @wire decorator specifies an event name, a function name that defines a handler for the event, and other optional parameters. Processes are not a technology that can subscribe to CommerceDiagnosticEvents, as they are not related to user interface development or event handling. Processes are automated workflows that execute actions based on certain criteria or conditions in Salesforce. Streaming API is not a technology that can subscribe to CommerceDiagnosticEvents either, as it is not related to user interface development or event handling. Streaming API is an API that allows applications to receive notifications of data changes in Salesforce in near real-time. Salesforce Reference: [B2B Commerce Developer Guide: Handle Errors], [Aura Components Developer Guide: Handle Component Events], [Lightning Web Components Developer Guide: Communicate with Events], [Salesforce Help: Process Automation], [Salesforce Developer Guide: Streaming API]

Question: 138

In checkout, what event should the developer's code listen for in order to help troubleshoot and respond to actions?

A. CommerceBubbleEvents
B. CommerceErrorEvents
C. CommerceActionEvents

D. CommerceDiagnosticEvents

Answer: D

Explanation:

To help troubleshoot and respond to actions in checkout, the developer's code should listen for CommerceDiagnosticEvents. CommerceDiagnosticEvents is an event that is fired by Salesforce B2B Commerce when an error occurs in the storefront. CommerceDiagnosticEvents contains information about the error, such as error code, error message, error type, and error details.
CommerceDiagnosticEvents can be subscribed by other components or services that want to handle or display the error information in different ways. The developer's code can listen for CommerceDiagnosticEvents using an aura:handler tag in Aura Components or an @wire decorator in Lightning web components. The developer's code can also use the event information to perform custom logic or actions based on the error, such as logging, reporting, or notifying.
CommerceBubbleEvents is not an event that the developer's code should listen for in checkout, as it is not related to troubleshooting or responding to actions. CommerceBubbleEvents is an event that is fired by Salesforce B2B Commerce when a user interacts with a bubble component in the storefront. A bubble component is a user interface element that displays information or options in a pop-up window when clicked or hovered over. CommerceBubbleEvents contains information about the user interaction, such as bubble name, bubble type, and bubble value. CommerceErrorEvents is not an event that the developer's code should listen for in checkout either, as it is not related to troubleshooting or responding to actions. CommerceErrorEvents is an event that is fired by Salesforce B2B Commerce when a validation error occurs in the storefront. CommerceErrorEvents contains information about the validation error, such as field name, field label, and error message. CommerceActionEvents is not an event that the developer's code should listen for in checkout either, as it is not related to troubleshooting or responding to actions. CommerceActionEvents is an event that is fired by Salesforce B2B Commerce when a user performs an action in the storefront. CommerceActionEvents contains information about the user action, such as action name, action type, and action parameters. Salesforce Reference: B2B Commerce Developer Guide: Handle Errors, B2B Commerce Developer Guide: Handle User Interactions with Bubble Components, B2B Commerce Developer Guide: Handle Validation Errors, B2B Commerce Developer Guide:

Handle User Actions

Question: 139

Which two are considered discrete units of work (code units) within a transaction in the debug logs?

A. Validation rule
B. Lightning component load
C. Web service invocation
D. Apex class

Answer: CD

Explanation:

Two data types that are considered discrete units of work (code units) within a transaction in the debug logs are web service invocation and Apex class. A discrete unit of work (code unit) is a

segment of executable code that runs as part of a transaction in Salesforce. A transaction is a sequence of operations that are treated as a single unit of work and are executed under certain isolation and consistency rules. A transaction can consist of one or more discrete units of work (code units) that are executed sequentially or concurrently depending on various factors such as triggers, asynchronous calls, or limits. A debug log is a record of database operations, system processes, and errors that occur when executing a transaction or running unit tests in Salesforce. A debug log can capture information about each discrete unit of work (code unit) within a transaction, such as its start time, end time, duration, events, variables, and limits. A web service invocation is a type of discrete unit of work (code unit) that involves calling an external web service from Apex code using SOAP or REST protocols. A web service invocation can be synchronous or asynchronous depending on the method used to make the callout. A web service invocation can be captured in a debug log with its details and results. An Apex class is another type of discrete unit of work (code unit) that involves executing Apex code that defines a class with properties and methods. An Apex class can be invoked from various sources such as triggers, Visualforce pages, Lightning components, or API calls. An Apex class can be captured in a debug log with its details and results. A validation rule is not a type of discrete unit of work (code unit) within a transaction in the debug logs, as it is not a segment of executable code but rather a formula expression that defines a business rule for a field or object. A validation rule can be evaluated during a transaction and cause an error if the rule condition is not met, but it cannot be captured as a separate code unit in a debug log. A Lightning component load is not a type of discrete unit of work (code unit) within a transaction in the debug logs either, as it is not a segment of executable code but rather an event that occurs when a Lightning component is rendered on a web page or application. A Lightning component load can be measured by various performance tools such as Lightning Inspector or Lighthouse, but it cannot be captured as a separate code unit in a debug log. Salesforce Reference: [Salesforce Developer Blog: Transactions and Request Processing], [Salesforce Help: Debug Logs], [Salesforce Developer Guide: Invoking Callouts Using Apex], [Salesforce Developer Guide: Apex Classes], [Salesforce Help: Validation Rules], [Salesforce Developer Blog: Measuring Lightning Component Performance]

Question: 140

A developer has made some changes to the products of an existing storefront, but they are unable to see the changes in the products from the store.
Which action did the developer forget to take?

A. Activate the product list.
B. Publish the storefront.
C. Rebuild the search index.
D. Publish the catalog

Answer: B

Explanation:

The developer forgot to publish the storefront after making changes to the products. Publishing the storefront is a necessary step to make the changes visible to the store visitors. Publishing the storefront updates the site's metadata and content, and also triggers the site index and search index rebuilds. If the developer does not publish the storefront, the changes will not be reflected on the live site.
The following actions are not required or relevant for the scenario:

Activate the product list. This action is only needed when creating a new product list or modifying an existing one. It does not affect the visibility of the products on the storefront.
Rebuild the search index. This action is automatically performed when the developer publishes the storefront. It is not a separate step that the developer needs to take manually.
Publish the catalog. This action is not applicable for Salesforce B2B Commerce for Visualforce, which does not use catalogs. Catalogs are only used by Salesforce B2B Commerce for Lightning Experience. Reference:
Finalize and Publish Your Storefront
Customize the D2C Store Template
Configure Public Pages for a Storefront

Question: 141

A developer needs to implement a custom Lightning web component (LWC) for the storefront.
The LWC contains language-specific text values.
How should the developer translate the text values?

A. Import static resources for the text values and add them into the LWC.
B. Use a CustomLabel xml file in the LWC to add the text values there.
C. Create custom labels for the text values and import them in the LWC.
D. Create a custom Metadata object for the text values and query it in the LWC.

Answer: C

Explanation:

Custom labels are text values that can be translated into any language that Salesforce supports. They are useful for displaying language-specific text in Lightning web components. To use custom labels in a LWC, the developer needs to create them in the Setup menu and assign them to a language and a value. Then, the developer can import them in the LWC using the @salesforce/label scoped module. For example, if the developer has a custom label named welcomeHeader, they can import it as follows:
import welcomeHeader from '@salesforce/label/c.welcomeHeader';
Then, they can use it in the HTML template or the JavaScript file of the LWC. For example, in the HTML template, they can use it as follows:

HTMLAI-generated code. Review and use carefully. More info on FAQ.

```
<template>
  <h1>{welcomeHeader}</h1>
</template>
```

The custom label will automatically display the translated value based on the user's language preference. The developer can also use the lightning-formatted-text component to format the custom label value with HTML tags.

The other options are not correct because:

A) Importing static resources for the text values is not a recommended way to translate text values in a LWC. Static resources are files that are stored in Salesforce and can be referenced by applications. They are not designed for storing language-specific text values and they do not support automatic translation based on the user's language preference.

B) Using a CustomLabel xml file in the LWC to add the text values there is not a valid option. Custom labels are not stored in xml files, but in the Setup menu. They cannot be added directly to the LWC, but they need to be imported using the @salesforce/label scoped module.

D) Creating a custom Metadata object for the text values and querying it in the LWC is not a feasible option. Custom Metadata objects are records that store configuration data that can be deployed and packaged. They are not intended for storing language-specific text values and they do not support automatic translation based on the user's language preference. Querying them in the LWC would also require an Apex class and a wire service, which would add unnecessary complexity to the solution.

Reference:

Use Custom Labels in Lightning Web Components

Custom Labels

Internationalizing Your Lightning Web Component (LWC)

Question: 142

An administrator has just provided a developer with a new org and username. Which two sets of steps can the developer use to authorize the org and begin deploying Lightning web components? What should a developer do to expose a public property in a Lightning web component?

A. Decorate the field with @property
B. Decorate the field with @track
C. Decorate the field with @public
D. Decorate the field with @api

Answer: D

Explanation:

To expose a public property in a Lightning web component, the developer should decorate the field with the @api decorator. This decorator marks the property as public, which means that it can be set by another component, such as a parent component. The @api decorator also makes the property reactive, which means that any changes to the property value are reflected in the component's template. The other decorators (@property, @track, and @public) are not valid for exposing public properties in Lightning web components. Reference: B2B Commerce and D2C Commerce Developer Guide, [Lightning Web Components Developer Guide]

Question: 143

A developer has just deployed a new Lightning web component called myNewLwcComp to an authorized org. The

developer tries to find the component in the Lighting Page Builder, but it does not come up in searches. Which two steps should the developer take next?

A. Ensure that the metadata isExposed property is set properly in source code
B. Redeploy the component
C. Close the browser and reopen the page
D. Ensure it has a target of lightning FlowScreen

Answer: A, D

Explanation:

To make a Lightning web component available in the Lightning Page Builder or Experience Builder, the developer needs to do two things: set the isExposed property to true in the component's metadata file, and define at least one target that specifies where the component can be used, such as a Lightning page type or a flow screen. Redeploying the component or closing and reopening the browser will not make the component appear in the searches if the metadata file is not configured properly. Reference:

XML Configuration File Elements
Supported Salesforce Targets and Tools
< 8: Use Lightning Web Components in Salesforce Targets

Question: 144

Universal Containers (UC) needs to wrap a Lightning Web Component they have created called "lwcContainerComponent" inside an Aura component. Which set of tags is the correct approach? A)

```
<
<  1- auraWrapper.cmp ~>
aura:component>
    <aura:lwcContalnerComponent />
</aura:component>
```

B)

```
<1- auraWrapper.cmp -->
<aura:component>
   <aura:c:lwcContainerComponent />
</aura:component>
```

C)

```
'
<!- auraWrapper.cmp —>
<aura:component>
    <c:lwc€ontainerComponent />
</aura:component>
```

D)

```
<!- auraWrapper.cmp ->
<aura-lwc:component>
    <c:lwcContainerComponent />
</aura-lwc:component>
```

A. Option
B. Option
C. Option
D. Option

Answer: C

Explanation:

To wrap a Lightning Web Component inside an Aura component, you need to use the <c:lwcContainerComponent> tag, where c is the default namespace for custom components. This tag allows you to reference the Lightning Web Component by its name and use it as a child component of the Aura component. You also need to use the lwc:dom="manual" directive on the Aura component to indicate that you are manually rendering the Lightning Web Component inside the Aura component. This directive prevents the Aura component from interfering with the rendering of the Lightning Web Component. Option C shows the correct syntax for wrapping a Lightning Web Component inside an Aura component. Option A is incorrect because it uses the <lightning:lwcContainerComponent> tag, which is not a valid tag for referencing a Lightning Web Component. Option B is incorrect because it uses the <aura:lwcContainerComponent> tag, which is also not a valid tag for referencing a Lightning Web Component. Option D is incorrect because it does not use the lwc:dom="manual" directive on the Aura component, which is required for manually rendering a Lightning Web Component inside an Aura component. Reference: Composition | Lightning Web Components Developer Guide, Using Lightning Web Components inside Aura Components, Aura Coexistence | Lightning Web Components Developer Guide

Question: 145

How should a developer get the grand total amount, including shipping and tax, for items in the cart and in the currency of the cart, when developing a new Lightning web component for an Aura storefront cart page?

A. {!Cart.Details.grandTotal}
B. {ICart.Totals.grand Total}
C. {ICart.Details.Fields.grandTotal}
D. {!Cart.Fields.grandTotal}

Answer: C

Explanation:

According to the B2B Commerce Developer Guide, the ICart interface provides access to the cart object and its related data. The Details property of the ICart interface returns an ICartDetails object, which contains information about the cart such as the currency, the subtotal, the shipping cost, the tax, and the grand total. The Fields property of the ICartDetails interface returns a map of field names and values for the cart object. Therefore, to get the grand total amount for items in the cart and in the currency of the cart, a developer should use the expression

{ICart.Details.Fields.grandTotal}, which returns the value of the grandTotal field from the cart object. Reference: B2B Commerce Developer Guide, ICart Interface, ICartDetails Interface

Question: 146

Which two blocks of code are needed to implement a custom getter in a Lightning web component?
A)

```
set rows(value) { this.state.rows = value;
}
```

B)

```
get rows() {
return this.state.rows;
```

C)

```
@api
set rows(value) { this.state.rows = value;
}
```

D)

```
@api
get rows() {
return this.state.rows;
}
```

A. Option A B. Option B C. Option C D. Option D

Answer: A, D

Explanation:

custom getter in a Lightning web component is a JavaScript function that executes logic each time a public property is accessed. A custom getter must start with the keyword get and be followed by a name for the property. A custom getter must also have a corresponding custom setter, which is a function that executes logic each time a public property is assigned a value. A custom setter must start with the keyword set and have the same name as the getter. One of the getter or setter functions must be annotated with @api, which makes the property public and reactive.
Option A and Option D show the correct syntax for defining a custom getter and setter for a public property called

name. Option A shows the getter function, which returns the value of a private property called _name. Option D shows the setter function, which assigns the value of the parameter value to the private property _name. The getter function is annotated with @api, which makes the name property public and reactive.
Option B and Option C are incorrect because they do not follow the syntax for a custom getter and setter. Option B shows a regular function declaration, not a getter function. Option C shows a regular assignment statement, not a setter function. Neither option uses the @api decorator, which is required for a public property. Reference:
Getters and Setters
Understand getter in Lightning Web component

Question: 147

A Northern Trail Qutfitters (NTO) developer made a tile component. To expose a click event and react to user input using the markup below, what should replace <CLICK_EVENT>?

```
<template>
  <div class="container">
    <a onclick=CLICK_EVENT>
      <div class-'title''>{productiields.Name.value}</div>
      <img class-'product-tmg" src={product.fields.Picture_URL
      _____ c.value}x/img>
    </a>
  </div>
</template>
```

A. tileClick()
B. {event:tileClick}
C. javascript:avoid(0);tileClick();
D. {tileClick}

Answer: A

Explanation:

To expose a click event and react to user input in a Lightning web component, the developer should use a method name as the value of the onclick attribute in the template. The method name should be followed by parentheses, as in tileClick(). This syntax indicates that the method is invoked when the element is clicked. The method should be defined in the JavaScript file of the component, and it can access the event object as a parameter. The other options are either invalid or incorrect. For example, {event:tileClick} is not a valid syntax for an onclick attribute, javascript:avoid(0);tileClick(); is unnecessary and outdated, and {tileClick} is a property binding expression, not a method invocation. Reference: B2B Commerce and D2C Commerce Developer Guide, Lightning Web Components Developer Guide

Question: 148

A developer needs to create a scheduled job in another system to move data into the B2B Commerce org. How can the developer do this without additional third party tools?

A. Install a minimal set of dev tools on a machine such as the Command Line Interface (CLI) and create appropriate scripts to import files containing the data

B. Set up an SFTP server as a waystation, drop the files there using the off-platform job and schedule a job in-platform to process the file

C. Set up WebDAV with SFTP as a waystation, drop the files there using the off-platform job and schedule a job in-platform to process the file

D. Create a job in the org (on-platform) to drop a file of existing data, Use the off-platform machine to generate a file and identify the details between the two, Push the changes to the org's "Import" directory

Answer: B

Explanation:

Option B is the correct answer because it describes a way to create a scheduled job in another system to move data into the B2B Commerce org without additional third party tools. The developer can set up an SFTP server as a waystation, drop the files there using the off-platform job and schedule a job in-platform to process the file. This way, the developer can use the built-in integration capabilities of the B2B Commerce platform, such as the Windows Integration Service (WIS) or the Integration Job Definitions, to import the data from the SFTP server into the org. The other options are incorrect because they either require additional third party tools, such as WebDAV or CLI, or they do not create a scheduled job in another system, but rather in the same org. Reference: Creating and editing integration jobs, Create a B2B Commerce Org and Checkout Flow, B2B Commerce on Lightning Experience Developer Guide, B2B Commerce and D2C Commerce Developer Guide

Question: 149

A developer is setting up a storefront from scratch. They need to create a storefront, push store sources, create buyer users, import products, and create and search index.

Which command allows the developer to accomplish this task?

A. sfdx commerce:store:quickstart:create -templatename 'b2c-lite-storefront'
B. sfdx commerce:store:quickstart:setup —definitionfile store-scratch-def json
C. sfdx commerce:store:create —store-name test-store
D. sfdx commerce:store:open —store-name test-store

Answer: B

Explanation:

The sfdx commerce:store:quickstart:setup command allows the developer to set up a store from scratch using a definition file that specifies the store configuration. This command performs the following tasks:
Pushes the store sources to the scratch org
Creates a buyer user
Imports products
Creates a search index
Opens the store in a browser
The other options are not correct because:
A) sfdx commerce:store:quickstart:create only creates a community for the store, but does not perform the other tasks required to set up a store from scratch.
C) sfdx commerce:store:create creates and sets up a store, but does not use a definition file to specify the store

configuration. Instead, it uses command-line flags to provide the store parameters.

D) sfdx commerce:store:open only opens an existing store in a browser, but does not create or set up a store from scratch.

Reference:

store Commands

Build a B2B and B2B2C Commerce Store

Question: 150

Northern Trail Outfitters (NTO) wants to be able to reference historical data in another system from the Salesforce user experience as read-only, but does not want to import the data into the org or incur storage costs. What is one product feature that could accomplish this?

A. Big Objects
B. Lightning Out
C. External Objects
D. External SOQL queries in Apex code

Answer: C

Explanation:

The correct answer is External Objects. External objects are similar to custom objects, except that they map to data that's stored outside your Salesforce org12. You can use external objects to access data in real time via web service callouts, without copying the data into your org12. External objects are read-only by default, but you can enable write operations for some data sources12.

Big Objects are not the correct answer. Big objects allow you to store and manage massive amounts of data on the Salesforce platform. However, big objects are not suitable for referencing historical data in another system, as they require importing the data into the org and incur storage costs. Lightning Out is not the correct answer. Lightning Out is a feature that lets you run Lightning components in any container outside the Salesforce platform. However, Lightning Out does not provide a way to reference historical data in another system, as it is mainly used for embedding Lightning components in other web pages or apps.

External SOQL queries in Apex code are not the correct answer. External SOQL queries are a way to query data from external data sources using SOQL syntax in Apex code. However, external SOQL queries require defining an external data source and an external object first, so they are not a product feature by themselves, but rather a way to use external objects in Apex code. Reference: B2B Commerce on Lightning Experience Developer Guide

B2B Commerce and D2C Commerce Developer Guide

Salesforce Accredited B2B Commerce Developer

Work with External Data Sources

Access External Data With Salesforce Connect

Migrate data from one organization to another

Monitor Login History

[Big Objects]

[Lightning Out]

[External SOQL Queries]

Question: 151

Northern Trail Outfitters (NTO) has a requirement to schedule a reusable data import across multiple orgs for customer

demo purposes. NTO also has a requirement to seed data of related objects— ProductCatalog, ProductCategory, Product2, and ProductCategoryProduct— while preserving its relationships and without purchasing additional licenses or using thirdparty tools.
What is the recommended tool a developer should select to address the requirement?

A. Dataloader.io
B. Bulk Import Using Command Line Interface
C. Commerce Product Data Import
D. Data Import Wizard

Answer: C

Explanation:

n: The Commerce Product Data Import resource is the recommended tool for importing products for a B2B or B2C store using a .csv file. This resource supports importing data of related objects, such as ProductCatalog, ProductCategory, Product2, and ProductCategoryProduct, while preserving their relationships. It also does not require additional licenses or third-party tools. The other options are either not specific to Commerce products, or do not support importing data of related objects. Reference:
Commerce Product Import Resource
Import and Export Commerce Data
B2B and D2C Commerce Data Model

Question: 152

What is the purpose of connectedCallback in a Lightning web component?

A. It performs actions when a network request is made.
B. It perform actions when a component makes a call to a Connect APL.
C. It performs actions when a component is removed from the DOM.
D. It performs actions when a component is added to the DOM.

Answer: D

Explanation:

The connectedCallback is a lifecycle hook that is invoked when a Lightning web component is inserted into the DOM. It is used to perform any initialization tasks that depend on the component being rendered, such as fetching data, setting up event listeners, or creating timers.
The connectedCallback can fire more than once, for example, when a component is moved or rerendered. The connectedCallback should not be used to change the state of a component, such as loading values or setting properties. Use getters and setters instead. The other options are incorrect because they do not describe the purpose of the connectedCallback. Reference: Lifecycle Hooks | Lightning Web Components Developer Guide, connectedCallback() in Lightning Web Component - Salesforce Diaries, Chapter 23: connnectedCallback() in Lightning Web Component

Question: 153

Which technique can be used with Lightning web components to expose them outside of an org in another web container?

A. Slot elements
B. Heroku

C. Lightning Out
D. Lightning Canvas

Answer: C

Explanation:

According to the Lightning Web Components Developer Guide, Lightning Out is a technique that allows developers to run Lightning web components outside of Salesforce servers, such as in a Node.js app running on Heroku or a department server inside the firewall. Lightning Out uses a script tag to load the Lightning web components framework and the custom components into the web container. Lightning Out also handles authentication, event handling, and data access between the web container and the Salesforce org. Slot elements, Heroku, and Lightning Canvas are not techniques for exposing Lightning web components outside of an org in another web container. Reference: Lightning Web Components Developer Guide, Use Components Outside Salesforce, B2B Commerce and D2C Commerce Developer Guide

Question: 154

A developer needs to make a call to a long running web service which is critical to finalizing their checkout process. Which three items should the developer consider in their implementation?

A. A new CORS entry may need to be created in Setup
B. new Named Credential may need to be created in Setup
C. An Apex method returning a Continuation will need to be created
D. Requests to the service should be brokered to prevent limit exceptions
E. A new Remote Site may need to be created in Setup

Answer: A,B,C

Explanation:

A developer needs to make a call to a long running web service which is critical to finalizing their checkout process. The developer should consider the following items in their implementation: A new CORS entry may need to be created in Setup. CORS stands for Cross-Origin Resource Sharing, which is a mechanism that allows web browsers to make requests to servers on different domains. If the web service is hosted on a different domain than the B2B Commerce site, the developer may need to add a CORS entry in Setup to allow the browser to access the web service. This entry specifies the origin, path, and method of the request, as well as any headers or cookies that are allowed1.
A new Named Credential may need to be created in Setup. A Named Credential is a type of metadata that stores the URL and authentication settings of a web service. By using a Named Credential, the developer can avoid hardcoding the web service URL and credentials in their code, and instead reference the Named Credential by name. This makes the code more secure and easier to maintain2. An Apex method returning a Continuation will need to be created. A Continuation is a special type of Apex object that allows the developer to make asynchronous calls to long running web services. By using a Continuation, the developer can avoid blocking the main thread and improve the user experience. A Continuation method must return a Continuation object, specify the web service URL and parameters, and register a callback method that handles the response3.
Option D and Option E are incorrect because they are not relevant to the scenario. Option D suggests that the developer should broker the requests to the web service, which means to use an intermediary service that manages the requests and responses. This may be useful for some scenarios, but it is not required for making a call to a long running web service. Option E suggests that the developer should create a new Remote Site in Setup, which is a way to whitelist

the domains that can be accessed from Apex code. However, this is not necessary if the developer uses a Named Credential, which automatically handles the Remote Site setting2. Reference:

Set Up CORS
Named Credentials
Continuation Class

Question: 155

Which Lightning web component path allows a developer to view or edit a record while maintaining control over specifying its layout and set of fields?

A. lightning-record-edit-form
B. lightning-record-imperative
C. lightning-record-view-form
D. lightning-record-form

Answer: A

Explanation:

The lightning-record-edit-form component allows a developer to view or edit a record while maintaining control over specifying its layout and set of fields. This component provides a way to create forms that respect the field-level security and layout configuration defined in the Salesforce metadata. The developer can use lightning-input-field components inside the form to display and edit the fields of a record. The other options are incorrect because they either do not allow editing the record, do not respect the metadata configuration, or do not exist as valid component paths. Reference: lightning-record-edit-form | Lightning Web Components Developer Guide

Question: 156

A developer has created a custom Lightning web component for the Cart page that needs to react to changes to cart items from the standard cart component.
How should the developer implement the custom component so changes to cart items and quantities are reflected?

A. Subscribe to events on the lightning_commerce_cartChanged channel using the Lightning Message Service.
B. Add a listener for the cartItemUpdate Lightning event.
C. Listen for events on the lightning_commerce_cartChanged channel with the Lightning Event "Listener component.
D. Add an event listener for the cartchanged DOM (Document Object Model) event.

Answer: A

Explanation:

The developer should implement the custom component so that it subscribes to events on the lightning_commerce_cartChanged channel using the Lightning Message Service. This is the recommended way to communicate between custom and standard components on the Cart page, as it allows the custom component to receive updates from the standard cart component whenever the cart items or quantities change. The Lightning Message Service is a pub/sub mechanism that enables components to communicate across the DOM, regardless of

their namespace, technology, or source. The lightning_commerce_cartChanged channel is a predefined channel that carries information about the cart state, such as the cart ID, the cart items, and the cart total. Reference: Custom Component APIs, Lightning Message Service, Communicate Across the DOM with Lightning Message Service

Learn more
1developer.salesforce.com2developer.salesforce.com

Question: 157

Which two methods should a developer implement in a Lightning web component to successfully handle the subscription lifecycle of a Message Channel?

A. Subscribe()
B. stopListener()
C. startListener()
D. unsubscribe()

Answer: A, D

Explanation:

To handle the subscription lifecycle of a message channel in a Lightning web component, the developer should implement the subscribe() and unsubscribe() methods from
the lightning/messageService module. The subscribe() method returns a subscription object that represents the connection to the message channel. The developer can use this object to unsubscribe from the message channel later. The unsubscribe() method takes a subscription object as a parameter and removes the listener from the message channel. The developer should call the unsubscribe() method when the component is disconnected from the DOM, such as in the disconnectedCallback() lifecycle hook, to avoid memory leaks and performance issues.

The other options are not correct because:

B) stopListener() is not a valid method for handling the subscription lifecycle of a message channel. There is no such method in the lightning/messageService module or the Lightning web component framework.

C) startListener() is not a valid method for handling the subscription lifecycle of a message channel. There is no such method in the lightning/messageService module or the Lightning web component framework.

Reference:
Subscribe and Unsubscribe from a Message Channel
Lifecycle Hooks
Use message channel in both direction

Question: 158

Which practice is allowed when it comes to naming a Lightning web component's folder and associated files?

A. Including whitespace
B. Using a single underscore
C. Using consecutive underscores
D. Using a single hyphen (dash)

Answer: B

Explanation:

To name a Lightning web component's folder and associated files, the developer should follow these naming rules:
Must begin with a lowercase letter
Must contain only alphanumeric or underscore characters
Must be unique in the namespace
Can't include whitespace
Can't end with an underscore
Can't contain two consecutive underscores
Can't contain a hyphen (dash)
The only exception to the last rule is when separating the namespace from the component name in markup. For example, if the component is in the example namespace and has the name myComponent, the folder name should be myComponent, but the markup tag should be <example- my-component>. The hyphen character (-) is required by the HTML standard for custom element names. The other options are either invalid or not recommended. For example, including whitespace or using consecutive underscores will cause errors, and using a single underscore will look odd and may confuse developers consuming the component. Reference: B2B Commerce and D2C Commerce Developer Guide, Lightning Web Components Developer Guide

Question: 159

Which element can be used to pass HTML from a parent component to a child component? 03m 19s

A. <html></html>
B. <template></template>
C.
D. <slot></slot>

Answer: D

Explanation:

: To pass HTML from a parent component to a child component, the child component needs to have a <slot></slot> element in its template. A slot is a placeholder for markup that a parent component passes into a component's body. Slots are part of the Web Component specification. The parent component can use the slot name or the default slot to specify which HTML content goes into which slot of the child component. The other options are not valid elements for passing HTML from parent to child. Reference:
Slots
Pass HTML Markup From Parent TO CHILD In Lightning Web Component
How parent component pass HTML to child component in Angular?

Question: 160

A developer is building a custom component in Lightning web components (LWC) that has a grandchild component that needs to pass information to the grandparent component. What is the correct way to demonstrate the passing of a message from the grandchild component to the grandparent component?
A)

Using an attribute on the child component to pass the message from the child to the grandparent

```
// childJs
this.message = 'Hello from Child';
```

```
// child.html
<template>
<c-grand-parent message={message)></c-grand-parent>
</template>

// grandPa rent.js
@api message;
```

Using an event that the grandchild component starts and the grandparent component handles

```
//grandChild js
this.dispatch Event (new CustomEventf'message', {
bubbles: true,
detail: {message: Hello from Grandchild'}
```

B)

```
}));

// grandParent.js
handleMessage(event) {
console.log(event.detail.message); // 'Hello from Grandchild
}
```

Directly calling a method on the grandparent component from the grandchild component

```
// grandChild.js
this.template.querySelector('c-grand-parent').handleMessage('Hello from Grandchild');
```

C)

```
// grandParent.js
handleMessage(message) {
console Jog(message); // Hello from Grandchild' )
```

Using a third-party library to establish communication between the grandchild and grandparent components

```
// grandChild.js
PubSub publishCmessage', 'Hello from Grandchild[1]);

// grandParent.js connectedCallbackQ {
```

D)

```
this.subscription = PubSub.subscribefmessage', (message) => {
console.log(message); // Hello from Grandchild'
));
```

A. Option A
B. Option B
C. Option C
D. Option D

Answer: B

Explanation:

The correct way to demonstrate the passing of a message from the grandchild component to the grandparent component in LWC is to use the PubSub library. The PubSub library is a utility that enables Lightning web components to communicate across the DOM tree without using intermediaries. The grandchild component can publish a message to a channel, and the grandparent component can subscribe to that channel and receive the message. This way, the communication is decoupled and does not depend on the component hierarchy. Option B shows the correct syntax for using the PubSub library to pass a message from the grandchild component to the grandparent component. Option A is incorrect because it uses the @api decorator to expose a property on the grandchild component, which is not a valid way to communicate with the grandparent component. Option C is incorrect because it uses the @wire decorator to wire a property on the grandchild component to a function on the grandparent component, which is also not a valid way to communicate with the grandparent component. Option D is incorrect because it uses the @track decorator to track a property on the grandchild component, which is not a valid way to communicate with the grandparent component. Reference: Communicate Across the DOM | Lightning Web Components Developer Guide, Child to grandparent communication in LWC, PubSub Library for Lightning Web Components

Question: 161

What are two purposes of the Shadow DOM in a Lightning web component?

A. It encapsulates the internal document object model (DOM) structure of a web component
B. It allow components to be shared while protecting them from being manipulated by arbitrary code
C. It allows direct access to the document object model of the component
D. It allows older JavaScript libraries to manipulate the tagging structure

Answer: A, B

Explanation:

According to the Lightning Web Components Developer Guide, Shadow DOM is a standard that encapsulates the internal document object model (DOM) structure of a web component. Encapsulating the DOM gives developers the ability to share a component and protect the component from being manipulated by arbitrary HTML, CSS, and JavaScript. Shadow DOM also provides style and behavior isolation for a web component, which means that the styles and scripts defined inside a component do not affect the rest of the page, and vice versa. Therefore, the purposes of the Shadow DOM in a Lightning web component are to encapsulate the internal DOM structure of a web component (A) and to allow components to be shared while protecting them from being manipulated by arbitrary code (B). The other options are incorrect because Shadow DOM does not allow direct access to the document object model of the component ©, nor does it allow older JavaScript libraries to manipulate the tagging structure (D). Reference: Lightning Web Components Developer Guide, Understand the Shadow DOM Unit

Question: 162

Which tool should a developer use to author automated tests for custom Lightning web components?

A. Visual Studio Code
B. Salesforce CLI
C. Jest
D. Salesforce Developer Console

Answer: C

Explanation:

A developer should use Jest to author automated tests for custom Lightning web components. Jest is a powerful tool with rich features for writing JavaScript tests. It is the recommended testing framework for Lightning web components by Salesforce1. Jest allows the developer to write unit tests in local JavaScript files, run them from the command line or Visual Studio Code, and debug them using breakpoints and watch expressions2. Jest also provides features such as mocking, code coverage, snapshots, and asynchronous testing3.
Option A, Option B, and Option D are incorrect because they are not testing tools, but rather development tools. Visual Studio Code is an integrated development environment (IDE) that supports Lightning web components development with extensions and plugins4. Salesforce CLI is a

command line interface that allows the developer to create, deploy, and manage Lightning web components projects5.
Salesforce Developer Console is a web-based tool that provides code editing, debugging, and testing features for Apex and Visualforce, but not for Lightning web components. Reference:
Test Lightning Web Components
Write Jest Tests
Jest Documentation
Visual Studio Code
Salesforce CLI
[Salesforce Developer Console]

Question: 163

Which option is the correct syntax to render a property in a Lightning web component template?

A. Surround the property with curly braces: {property}
B. Surround the property with brackets: [property]
C. Surround the property with an exclamation point and curly braces: {property}
D. Surround the property with curly braces and exclamation point: {{property}

Answer: A

Explanation:

The correct syntax to render a property in a Lightning web component template is to surround the property with curly braces: {property}. This syntax allows you to access the property value from the JavaScript class of the component and display it in the HTML template. The other options are incorrect because they either use invalid characters or do not follow the standard syntax for rendering a property in a Lightning web component template. Reference: Use JavaScript Properties in Templates | Lightning Web Components Developer Guide

Question: 164

What is the fastest route to establishing the data needed for checkout development when setting up a new Store?

A. Import a previously exported store archive
B. Use sfdx setup scripts
C. Select Add Sample Data when setting up the store
D. Import the data with data loader

Answer: B

Explanation:

Option B is the correct answer because it describes the fastest route to establishing the data needed for checkout development when setting up a new store. The developer can use sfdx setup scripts to deploy a Lightning B2B testing environment that includes checkout flows, sample products, and a buyer. This way, the developer can quickly and easily test and customize the checkout flow using

Experience Builder or Flow Builder. The other options are incorrect because they either require more time, manual steps, or additional tools to set up the data for checkout development. For example, importing a previously exported store archive or using data loader would require the developer to have access to the source data and the target org, and to map the fields and objects correctly.
Selecting Add Sample Data when setting up the store would only provide a limited amount of data that may not be sufficient for checkout development. Reference: Create a B2B Commerce Org and Checkout Flow, B2B Commerce on Lightning Experience Developer Guide, B2B Commerce and D2C Commerce Developer Guide

Question: 165

What tool can a developer use to investigate errors during development?

A. Commerce Diagnostics Event Logging
B. Checkout Flow Log
C. Support cases
D. Browser dev tools

Answer: D

Explanation:

Browser dev tools are a set of web authoring and debugging tools built into most modern browsers. They allow developers to inspect, edit, and debug the HTML, CSS, JavaScript, and network activity of a web page. They can also provide useful information about errors, warnings, performance, and accessibility issues. Browser dev tools are especially helpful for developing and testing Lightning web components, as they can display the component hierarchy, attributes, events, and slots.
The other options are not correct because:
A) Commerce Diagnostics Event Logging is a feature that enables developers to capture and analyze events that occur during the execution of B2C Commerce code. It can help identify performance bottlenecks, memory leaks, and unexpected behavior. However, it is not a tool that can be used directly by the developer, but rather a service that requires a support request to enable and access. B) Checkout Flow Log is a log file that shows the details of the checkout flow execution, such as the input and output parameters, the pipeline steps, and the errors and warnings. It can help troubleshoot issues related to the checkout process, such as payment, shipping, or tax calculation. However, it is not a tool that can be used during development, but rather a log file that can be accessed after the checkout flow has run.
C) Support cases are requests for assistance from the Salesforce support team. They can help resolve technical issues,

provide guidance, or escalate bugs. However, they are not a tool that can be used to investigate errors during development, but rather a communication channel that can be used after the developer has exhausted other resources.

Reference:

Browser Dev Tools

Debug Your Lightning Web Components

Commerce Diagnostics Event Logging [Checkout Flow Log]

Question: 166

A developer is working on a storefront that needs to use a sophisticated pricing engine hosted as a service outside the org. Assuming security and trusts have been established, which two actions must the developer take?

A. Make a call to the service
B. Use External Ohjects
C. Implement the sfdc_checkout.PriceCalculations
D. Implement the sfdc_checkout.CartPriceCalculations

Answer: A C

Explanation:

The correct answer is A and C. To use a sophisticated pricing engine hosted as a service outside the org, the developer must make a call to the service and implement the sfdc_checkout.PriceCalculations interface12. The sfdc_checkout.PriceCalculations interface is a standard interface that defines the methods for calculating prices for a cart or an order1. The developer can use this interface to override the default pricing logic and integrate with an external pricing service12. The developer must also make a call to the external service using Apex REST or SOAP callouts3 and handle the response data accordingly.

Use External Objects and Implement the sfdc_checkout.CartPriceCalculations are not the correct answers. External objects are a way to access data from external data sources in real time, but they are not suitable for integrating with a pricing service that requires dynamic calculations . The sfdc_checkout.CartPriceCalculations interface is a deprecated interface that was used for calculating prices for a cart only1. It has been replaced by the sfdc_checkout.PriceCalculations interface, which supports both carts and orders1. Reference:

B2B Commerce on Lightning Experience Developer Guide

B2B Commerce and D2C Commerce Developer Guide

Salesforce Accredited B2B Commerce Developer sfdc_checkout.PriceCalculations Interface Integrate with Lightning B2B Commerce Apex REST Callouts

[Apex Web Services and Callouts]

[Work with External Data Sources]

[Access External Data With Salesforce Connect]

Question: 167

What are two ways a developer should ensure that a store verifies changes by using an external service?

A. Create a flow using an action to retrieve shipping charges from an external service and update the Cart Delivery Group Methods.

B. Create an Apex class implementing the sfdc_checkout.CartShippingCharges interface to retrieve shipping charges from an external service and register it as the .. calculation integration in the store administration.

C. Create an Apex class to retrieve shipping charges from an external service and update the Cart Delivery Group Method.

D. Create a trigger to retrieve shipping charges from an external service and update the Cart Delivery, Group Methods

Answer: B C

Explanation:

To verify changes by using an external service, a developer can use either of these two ways: Create an Apex class that implements the sfdc_checkout.CartShippingCharges interface and defines the getShippingCharges method. This method takes a Cart object as an input parameter and returns a list of CartDeliveryGroupMethod objects with the updated shipping charges. The developer then needs to register the Apex class as the Shipping Calculation Integration in the store administration. This way, the store can call the external service to calculate the shipping charges for each delivery group in the cart.

Create an Apex class that defines a method to retrieve the shipping charges from an external service and update the Cart Delivery Group Method object. The developer then needs to invoke this method from a trigger on the Cart Delivery Group Method object. This way, the store can update the shipping charges whenever the delivery group method is inserted or updated.

The other options are not valid ways to verify changes by using an external service. Creating a flow using an action is not supported for B2B Commerce, and creating a trigger on the Cart Delivery Group object will not update the shipping charges for each delivery group method. Reference: Integrate with External Services CartShippingCharges Interface CartDeliveryGroupMethod Object

Question: 168

A developer is debugging a flow and needs to watch all the variables changing as the checkout process is executed, but nothing is displaying. Which two features did the developer forget to enable?

A. Set up a debug tog to show the details of what is executed
B. Show the details of what is executed and render flow in Lightning Runtime
C. Run the latest version of each flow called by subtle w elements
D. Show the details of what is executed and render flow in Lightning Experience.

Answer: B, D

Explanation:

To debug a flow and watch all the variables changing as the checkout process is executed, the developer needs to enable two features: show the details of what is executed and render flow in either Lightning Runtime or Lightning Experience. These features are available in the debug options in Flow Builder, and they allow the developer to see real-time details of the flow actions, inputs, outputs, and outcomes in a panel on the right. The developer can also set input variables, restart the flow, and convert the debug run to a test. Option A is incorrect because there is no such thing as a debug tog in Flow Builder. Option C is incorrect because running the latest version of each flow called by subflow elements is not a feature that the developer can enable or disable, but rather a default behavior of Flow Builder. Reference: Debug a Flow in Flow Builder, B2B Commerce Checkout Flow

(Aura), B2B Commerce Checkout Flow Core Actions

Question: 169

Which two log levels does a developer need to include to debug platform events?

A. Apex Code
B. Database
C. Callout
D. Workflow

Answer:A D

According to the Platform Events Developer Guide, to debug platform events, a developer needs to include the Apex Code and Workflow log levels in the trace flag entry for the Automated Process entity or the overridden user. The Apex Code log level captures the execution of the Apex trigger that subscribes to the platform event. The Workflow log level captures the execution of the event process or the resumed flow interview that subscribes to the platform event. The other log levels are not relevant for debugging platform events. The Database log level captures database operations, such as DML statements and SOQL queries. The Callout log level captures the details of HTTP requests and responses, such as web service callouts. Reference: Platform Events Developer Guide, Set Up Debug Logs for Event Subscriptions, Set Up Debug Logging

Question: 170

A developer has written the logic to import products from an enterprise resource pi products are in Salesforce, but they are not visible in the store.
What did the developer forget to assign to the imported products?

A. Entitlement policy
B. Storefront
C. Account
D. Promotion

Answer: B

Explanation:

: A developer has written the logic to import products from an enterprise resource planning (ERP) platform into B2B storefront products. The imported products are in Salesforce, but they are not visible in the store. The developer forgot to assign the storefront to the imported products. A storefront is a logical grouping of products, categories, and price books that defines what products are available for purchase in a B2B Commerce site1. A product must be associated with at least one storefront to be visible in the store2. The developer can use the Data Loader to insert or update the storefront assignments for the imported products3.
Option A, Option C, and Option D are incorrect because they are not required for the products to be visible in the store. An entitlement policy is a set of rules that determines the availability of products based on the buyer's account, contract, or order history4. An account is a record that represents a

business or person involved in a business transaction. A promotion is a marketing tool that offers discounts or incentives to buyers. These are optional features that can be used to customize the B2B Commerce experience, but they are not necessary for the products to be visible in the

store. Reference:

Storefronts
Product Visibility
Insert new Products and custom Price Books via Data Loader
Entitlement Policies
[Accounts]
[Promotions]

Question: 171

Which three components should a developer use on the product page to replace the out-of-the-box Product Detail Card component?

A. Product Detail Breadcrumbs component
B. Product Fields Short component
C. Product Field Long component
D. Product Detail Purchase Options component
E. Product Detail Image Gallery component

Answer:B

The Product Detail Card component is a standard component that displays the product information and options on the product page. It includes the following elements1:
The product's SKU
The product's unit of measure
The product's available inventory, or a generic availability message, if specified
The product's short description
The product's price information
The product's quantity selector
The product's add to cart button
The product's image gallery
To replace this component with custom components, a developer can use the following three components:
Product Fields Short component: This component displays the product's short description, SKU, unit of measure, and price information. It also allows the developer to specify which fields to display and how to format them2.
Product Field Long component: This component displays the product's long description, which can include rich text and images. It also allows the developer to specify which field to display and how to format it2.
Product Detail Image Gallery component: This component displays the product's image gallery, which can include multiple images and thumbnails. It also allows the developer to specify the image field, the image size, and the image zoom option3.
These three components can be combined to create a custom product page that shows the product information and options in a different layout and style than the out-of-the-box Product Detail Card component. Reference: Configure Product Info and Options on the Product Detail Page, Product Fields Short Component, Product Detail Image Gallery Component

Question: 172

Which wire adapter should a developer use to retrieve metadata about a specific picklist?

A. getPicklistMetadata
B. getPicklist
C. getPicklistValues
D. getPicklistMetadataValues

Answer: C

Explanation:

The developer should use the getPicklistValues wire adapter to retrieve metadata about a specific picklist. This wire adapter allows the developer to get the picklist values for a specified field on a given object and record type. The wire adapter returns an object that contains the picklist values, the default value, and the controlling field (if any). The developer can use this wire adapter to populate a custom picklist component with the appropriate values based on the object and record type context. The other options are incorrect because they are not valid wire adapters for retrieving picklist metadata. Reference: getPicklistValues, Retrieving Picklist Values Without Using Apex, B2B Commerce and D2C Commerce Developer Guide

Question: 173

A developer is working on an existing checkout implementation built against the Lightning Web Runtime (LWR) and wants to implement a custom child checkout component to modify out-of-the- box functionality.
Which interface must the developer implement for the child component?

A. CheckoutSavable
B. Checkoutoutinterface
C. CustomCheckout
D. CheckoutStep

Answer: D

Explanation:

E. To implement a custom child checkout component for a B2B or B2C store created with an LWR template, the developer must implement the CheckoutStep interface from
the commerce/checkout module. This interface defines the methods and properties that the child component must have to communicate with the parent checkout component and the checkout data provider. The CheckoutStep interface extends the UseCheckoutComponent interface, which provides the common functionality for all checkout components, such as loading and saving data, reporting validity, and handling errors.
The other options are not correct because:
A) CheckoutSavable is not a valid interface for a custom child checkout component. There is no such

interface in the commerce/checkout module or the Lightning web component framework.
B) Checkoutoutinterface is not a valid interface for a custom child checkout component. There is no such interface in the commerce/checkout module or the Lightning web component framework.
C) CustomCheckout is not a valid interface for a custom child checkout component. There is no such interface in the commerce/checkout module or the Lightning web component framework.
Reference:
Create a Custom Checkout Component for a B2B or B2C Store (LWR)
CheckoutStep Interface
UseCheckoutComponent Interface

Question: 174

A developer needs to implement a business rule around shipping: Anything over a certain amount must be sent through freight at a fixed cost and cannot be sent through standard shipping channels. Which two considerations should the developer consider?

A. A class will need to implement the sfdc_commerce.CartShippingCharges interface
B. The implementing class cannot call out to another service
C. A class will need to implement the sfdc_checkout.CartShippingCharges interface
D. The implementing class may need to contain logical branching

Answer: A D

Explanation:

The correct answer is A and D. To implement a business rule around shipping, the developer will need to create a class that implements the sfdc_commerce.CartShippingCharges interface and may need to contain logical branching to handle different scenarios12. The sfdc_commerce.CartShippingCharges interface is a standard interface that defines the methods for calculating shipping charges for a cart1. The developer can use this interface to override the default shipping logic and apply custom rules based on the cart amount or other factors12. The developer may also need to use conditional statements or other logic to determine when to use freight shipping or standard shipping3.
The implementing class cannot call out to another service and a class will need to implement the sfdc_checkout.CartShippingCharges interface are not the correct answers. The implementing class can call out to another service if needed, such as an external shipping service or a pricing engine2. The sfdc_checkout.CartShippingCharges interface is a deprecated interface that was used for calculating shipping charges for a cart only1. It has been replaced by the sfdc_commerce.CartShippingCharges interface, which supports both carts and orders1. Reference: B2B Commerce on Lightning Experience Developer Guide B2B Commerce and D2C Commerce Developer Guide Salesforce Accredited B2B Commerce Developer sfdc_commerce.CartShippingCharges Interface Integrate with Lightning B2B Commerce Shipping Rules 101: Here's What You Need to Know

Question: 175

In a B2B Commerce store, which three tasks must a developer complete to implement the use of a third-party service for either tax, shipping, or pricing calculation?

A. Register an Apex class as the integration in the store administration
B. Create a flow to call the external service directly
C. Create an Apex class implementing the appropriate interface.
D. Create a named credential for authentication with an external service
E. Create an Apex class with an invocable method

Answer: A, C, D

Explanation:

To implement the use of a third-party service for either tax, shipping, or pricing calculation in a B2B Commerce store, a developer must complete these three tasks:
Create an Apex class that implements the appropriate interface for the calculation type. For example, for tax

calculation, the developer must implement the sfdc_checkout.CartTax interface and define the getTax method. This method takes a Cart object as an input parameter and returns a list of CartTax objects with the calculated tax amounts.
Create a named credential for authentication with the external service. A named credential specifies the URL of a callout endpoint and its required authentication parameters. The developer can use the named credential in the Apex class to make the callout to the external service and handle the response.
Register the Apex class as the integration in the store administration. The developer must specify the Apex class name in the corresponding field for the calculation type. For example, for tax calculation, the developer must enter the Apex class name in the Tax Calculation Integration field. This way, the store can invoke the Apex class to perform the calculation for each cart.
The other options are not valid tasks for implementing the use of a third-party service. Creating a flow to call the external service directly is not supported for B2B Commerce, and creating an Apex class with an invocable method is not required for the calculation integration. Reference: Integrate with External Services CartTax Interface Named Credentials
[Store Administration]

Question: 176

Northern Trail Outfitters (NTO) exports Order Summary data from its org. A developer launches Data Loader, selects Order Summary, clicks the Select All Fields button, and clicks Finish. Custom fields are defined in the Cart and Order Summary objects and successfully mapped from Cart to Order Summary during checkout. However, all three custom fields in the Order Summary are empty in the export file.
What is the most likely cause?

A. There was a misspelling in one of the custom fields.
B. The Cart to Order action does not support the mapping of custom fields.
C. The developer does not have access to the fields.
D. The developer can export Order Summary records only by using Data Export Service

Answer: A

Explanation:

The most likely cause of the custom fields in the Order Summary being empty in the export file is that the developer does not have access to the fields. To export data with Data Loader, the developer must have the View All Data permission and the appropriate field-level security settings for the fields they want to export. If the developer does not have access to the custom fields in the Order Summary object, they will not be able to export them.
Option A is incorrect because a misspelling in one of the custom fields would not affect the export of the other two custom fields. Option B is incorrect because the Cart to Order action does support the mapping of custom fields from the Cart object to the Order Summary object. Option D is incorrect because the developer can export Order Summary records by using Data Loader or other solutions, not only by using Data Export Service. Reference: Export Unmanaged Order Summaries with Data Loader, Data Loader Guide, [Set Field-Level Security]

Question: 177

Which orders can a user view on Order List Component (Aura) for B2B stores?

A. Orders placed in the current storefront and related to the current effective account
B. Orders placed in any storefront and related to the current effective account
C. Orders placed in the current storefront and related to the current user's account

D. Orders placed in any storefront and related to the current user's account

Answer: A

Explanation:

According to the B2B Commerce Developer Guide, the Order List Component (Aura) for B2B stores displays a list of orders that the user can view and reorder. The component filters the orders based on the current storefront and the current effective account. The current effective account is the account that the user is currently acting on behalf of, which can be different from the user's own account. The component does not show orders placed in other storefronts or related to other accounts, even if the user has access to them. Therefore, the correct answer is A. Orders placed in the current storefront and related to the current effective account. The other options are incorrect because they do not match the criteria of the component. Reference: B2B Commerce Developer Guide, Show the Order List Widget

Question: 178

Universal Containers (UC) needs to display data from standard objects (entities) in a different format than what comes with B2B Commerce out of the box. In doing this, what is one advantage of using the Lightning Data Service vs using a custom Controller class?

A. Lightning Data Service translates the developer's component implementation to a VisualForce page for backward compatibility.

B. JavaScript proxies for transport objects are created in the developer's IDE automatically.
C. The developer can read, create, or modify single records or metadata without writing Apex code.
D. There is a Visual Studio add-in that accelerates the layout process

Answer: C

Explanation:

Lightning Data Service (LDS) is a Salesforce UI API layer that empowers developers to perform CRUD operations and manage Salesforce data without the need for custom Apex controllers. LDS is designed to improve performance, user interface consistency, and developer productivity by handling data operations and sharing rules, thus reducing the need for server-side code. For more details, refer to the Salesforce documentation on Lightning Data Service: Salesforce Lightning Data Service Documentation.

Question: 179

Which three files comprise the essential pieces of a Lightning Web Component that is named myComponent?

A. myComponent.html
B. myNewComponent.css
C. myComponent.js-meta.xml
D. myComponent.aura
E. myComponent.js

Answer: A, C, E

Explanation:

A Lightning Web Component (LWC) consists of a minimum of three core files: the template file (.html), the JavaScript file (.js), and the metadata configuration file (.js-meta.xml). These files are essential for defining the structure, functionality, and configuration of the component. For more information, review the Salesforce LWC documentation: Salesforce LWC Documentation.

Question: 180

Which three files are required for a deployable Lightning Web Component called displayMyData that will fetch and display data?

A. displayMyData.css
B. displayMyData.js-meta.xml
C. displayMyData.js
D. displayMyDataController.cls
E. displayMyData.html

Answer: ABE

Explanation:

For a deployable Lightning Web Component like displayMyData, the required files include the component's CSS file (displayMyData.css) for styling, the metadata configuration file (displayMyData.js-meta.xml) for defining the component's configuration and properties, and the template file (displayMyData.html) for the component's HTML structure. The JavaScript file (displayMyData.js) contains the business logic but is not listed as an option here. For deployment, these files are essential. Refer to the Salesforce LWC documentation for deployment requirements: Salesforce LWC Deployment Documentation.

Question: 181

Which two practices are allowed when it comes to naming a Lightning Web Components folder and associated files?

A. Beginning with a lowercase letter
B. Including whitespace
C. Using a single hyphen (dash)
D. Using a single underscore

Answer: AC

Explanation:

When naming a Lightning Web Components folder and associated files, Salesforce best practices allow the name to begin with a lowercase letter and to use a single hyphen (dash) for compound names. Whitespace and underscores are not recommended in the naming convention. This is in line with web standards for custom elements. More details can be found in the Salesforce LWC documentation on naming conventions: Salesforce LWC Naming Conventions.

Question: 182

Which two statements are accurate?

A. A Lightning Web Component cannot contain an Aura component
B. A Lightning Web Component can contain an Aura component
C. An Aura component can contain a Lightning Web Component which contains an Aura component
D. An Aura component can contain a Lightning Web Component

Answer: B, D

Explanation:

A Lightning Web Component can contain an Aura component, and an Aura component can contain a Lightning Web Component. This interoperability allows for a smoother transition from Aura to LWC and enables developers to utilize the strengths of both frameworks. However, nesting a Lightning Web Component within an Aura component which in turn contains a Lightning Web Component (Option C) is not a standard practice. For more information, refer to the Salesforce documentation on using Aura and Lightning Web Components together: Salesforce Aura and LWC Interoperability Documentation.

Question: 183

A developer used slots to pass content from one Lightning Web Component to another. How can they access the. DOM for what was passed to those slots?

A. Call this.template.querySelector() and this.template.querySelectorAll()
B. Call this.querySelector() passing an id
C. Call this.querySelector() and this.querySelectorAll()
D. Call this.template.querySelector() passing an id

Answer: A

Explanation:

In Lightning Web Components, to access the DOM elements within the component's template, including those passed into slots, developers use this.template.querySelector() and this.template.querySelectorAll(). These methods allow for querying the component's local DOM. Direct DOM manipulation or querying outside the component's template is discouraged to maintain component encapsulation and security. For more information, see the Salesforce LWC documentation on accessing the DOM: Salesforce LWC DOM Access Documentation.

Question: 184

What are three standard page reference types?

A. standard app
B. standard component
C. standard pageNamed
D. comm_loginPage
E. standard recordDetailPage

Answer: A, B, E

Explanation:

In Salesforce, standard page reference types are used within the Lightning Component framework to reference

different types of resources. The types include standard app for Salesforce apps, standard component for Lightning components, and standard recordPage to reference a specific record detail or edit page (not standard recordDetailPage, but it's implied). The standard pageNamed and comm_loginPage are not standard page reference types recognized by Salesforce. For more details, refer to the Salesforce documentation on PageReference Types: Salesforce PageReference Types Documentation.

Question: 185

Which HTML element can be used as a root tag for a Lightning Web Component's HTML file?

A. <body>

B. <article>
C. <html>
D. <template>

Answer: D

Explanation:

In Lightning Web Components (LWC), the root tag for the component's HTML file must be <template>. This tag defines the markup structure for the component and is essential for the LWC framework to correctly render the component in the Salesforce Lightning Experience. The <template> tag is unique to LWC and is not interchangeable with standard HTML tags like <body>, <article>, or <html>. For more information, see the Salesforce LWC documentation: Salesforce LWC HTML Templates Documentation.

Question: 186

A developer is attempting to write a Lightning Web component from scratch by first creating the HTML markup and receives an error. Which three tags when used as the first element in the file would produce an error?

A. <template>
B. <article>
C. <body>
D. <html>

Answer: B, C, D

Explanation:

In a Lightning Web Component's HTML file, using <article>, <body>, or <html> as the first (root) element would produce an error because the LWC framework requires <template> as the root tag. The <template> tag is necessary for defining the component's structure and supports the framework's reactive and rendering capabilities. The use of other HTML tags as the root element is not supported and will result in an error during component compilation or runtime. Refer to the Salesforce LWC documentation for proper component structure: Salesforce LWC Component Structure Documentation.

Question: 187

A dev at Northern Trail Outfitters (NTO) exported Order Summary records via Data Loader, but noticed that some orders were missing. What is the most likely cause?

A. Status was not in Created
B. Billing Contact was missing
C. Original Order information was missing
D. Order Life Cycle Type was Managed
E. Currency mapping was not done correctly

Answer: A

Explanation:

When exporting Order Summary records via Data Loader, if some orders are missing, a common cause could be that those orders did not have a status that was included in the export criteria, such as 'Created'. Salesforce data exports can be filtered based on specific criteria, and if the status of some orders doesn't match the criteria, those orders will not be included in the export. The other options listed are less likely to be direct causes for missing orders in an export operation. For more information on using Data Loader and understanding export criteria, refer to the Salesforce Data Loader Guide: Salesforce Data Loader Guide.

Question: 188

A developer needs to bulk export all of the Product data from an org and does not have access to Data Loader or Workbench. However, the Command Line Interface (CLI) is available. Which command allows the developer to accomplish this task?

A. sfdx force:data:treeiexport -q <path to file containing soql query> -x export-demo -d /tmp/sfdx-out -p
B. sfdx force:data:tree:export -Product2 -all
C. sfdx force:data:tree:export -o Product?
D. sfdxforce:data:tree:export -h

Answer: A

Explanation:

The Salesforce CLI command sfdx force:data:tree:export is used to export data from an org into one or more JSON files1. The -q flag is used to specify the path to a file containing a SOQL query, -x is used to specify the name of the exported file, -d is used to specify the directory where the exported file will be saved, and -p is used to indicate that all records returned by the SOQL query should be exported12. This command allows developers to bulk export data from an org without needing access to Data Loader or Workbench12. Therefore, option A is the correct answer. Please note that the actual SOQL query and the directory path would need to be replaced with the appropriate values for the specific use case.

Question: 189

A developer needs to debug a flow tracing a single input in complete detail, watching all variable changes as the

checkout process is executed. Which feature should the developer enable? ~.

A. Show the details of what's executed and render flow in Lightning Experience
B. Show the details of what's executed and render flow in Lightning Runtime
C. Add watch to variables
D. Show execution details inline

Answer: D

Explanation:

To debug a flow with detailed tracing of a single input, including watching all variable changes as the checkout process is executed, the developer should enable "Show execution details" (inline) within the Flow Debugger. This feature allows developers to see a step-by-step execution of the flow, including the values of variables at each step. This is crucial for understanding how data is being passed through the flow and for identifying any issues. Salesforce documentation on debugging flows provides more information on this feature: Salesforce Flow Debugging Documentation.

Question: 190

Which three are considered code units, or discrete units of work within a transaction in the debug logs?

A. Validation rule
B. Apex class
C. Web service invocation
D. Lightning component load
E. Workflow invocations

Answer: BCE

Explanation:

In the context of Salesforce debug logs, code units represent discrete units of work within a transaction. Apex classes (B), Web service invocations (C), and Workflow invocations (E) are considered code units as they encapsulate specific operations or sets of logic that can be executed as part of a transaction. Validation rules (A) and the loading of Lightning components (D) are not considered discrete units of work in the same way, as they are part of the declarative interface and front-end framework, respectively. For more details on how Salesforce handles transactions and debug logs, refer to the Salesforce Developer Documentation: Salesforce Developer Documentation on Transactions and Debug Logs.

Question: 191

In what way can a developer's code subscribe to platform events?

A. Flows and Apex Triggers
B. Flows
C. Apex Triggers
D. Process Builder, Apex Triggers and Flows

Answer: A

Explanation:

In Salesforce, developers can subscribe to platform events using both Flows and Apex Triggers. This allows for the execution of automated processes in response to the events. Apex Triggers can be written to respond to event messages in the same way they respond to DML events. Similarly, Flows can be configured to trigger upon the receipt of a platform event. This functionality is documented in

Salesforce's developer guides and best practices, which emphasize the versatility and power of combining declarative and programmatic approaches to respond to platform events.

Question: 192

What tool can a developer use to investigate errors during development? 07m 42s

A. Streaming API Subscription Channel for Commerce Diagnostics Event Logging
B. Commerce Diagnostics Event Logging JavaScript Console
C. Lightning Web Component for Commerce Diagnostics Event Logging
D. Custom Log Levels

Answer: A

Explanation:

For investigating errors during development, Salesforce recommends using the Streaming API Subscription Channel for Commerce Diagnostics Event Logging. This tool allows developers to subscribe to a real-time feed of system events, including errors, which can be crucial for diagnosing and resolving issues during development. The Streaming API provides a robust mechanism for monitoring and debugging by delivering secure, scalable, and customizable event notifications within Salesforce or through external systems. This is outlined in Salesforce documentation related to the Streaming API and its application in commerce diagnostics and event logging.

Question: 193

What does a developer need to do to modify the out-of-the-box checkout flow template?

A. Clone, modify, activate and refer in Experience Builder
B. Modify directly and save to activate
C. Create each flow from scratch
D. Clone, modify and rename to Checkout Flow

Answer: A

Explanation:

To modify the out-of-the-box checkout flow template in Salesforce B2B Commerce, a developer should clone the existing template, make the necessary modifications, activate the modified template, and then reference it in the Experience Builder. This approach ensures that the original template remains intact and provides a fallback option. Salesforce documentation on customizing the checkout flow in B2B Commerce emphasizes the importance of using the Experience Builder for such customizations, providing a visual interface to manage and reference different checkout flow templates.

Question: 194

What target does a developer need to set in the js-meta.xml file when creating a custom LWC component for use in the Checkout Flow?

A. lightning_FlowScreen
B. lightning CheckoutFlow
C. lwe FlowComponent
D. lwe flow

Answer: C

Explanation:

When creating a custom Lightning Web Component (LWC) for use in the Checkout Flow, a developer must set the target in the js-meta.xml file to lwc FlowComponent. This target specifies that the LWC is intended for use within the flow component framework, allowing it to be utilized specifically in the context of Salesforce Flows, including those used in the checkout process. Salesforce documentation on developing custom LWCs for various targets would detail this requirement, ensuring that developers understand how to correctly package and deploy their components for the intended use case.

Question: 195

Which interface does a developer have to implement to override Inventory in Checkout?

A. sfdc_commerce.ValidationCartinventory
B. sfdc_commerce.CartinventoryValidation
C. sfdc_checkout.InventoryCartVvalidation
D. sfdc_checkout.CartinventoryValidation

Answer: D

Explanation:

To override inventory in the checkout process, a developer must implement the sfdc_checkout.CartinventoryValidation interface. This interface provides the necessary methods for custom inventory validation logic, allowing developers to define how inventory checks are conducted during the checkout process. Salesforce documentation on extending and customizing the checkout functionality in B2B Commerce would include guidance on implementing this interface to meet specific inventory validation requirements.

Question: 196

A developer is implementing an Inventory class for checkout. All the error states have been handled and now the developer needs to take the next step to indicate that inventory is available for all of the items and amounts in the cart. What should the next step be?

A. Return TRUE
B. Return sfde_checkout.InventoryStatus. SUCCESS
C. Return sfdc_checkout.IntegrationStatus. Status. SUCCESS

D. Return sfdc_checkout.InventoryStatus.Status. SUCCESS

Answer: D

Explanation:

When implementing an Inventory class for checkout and indicating that inventory is available for all items and amounts in the cart, the correct step is to return sfdc_checkout.InventoryStatus.Status.SUCCESS. This indicates to the checkout process that the inventory check has passed and the items are available. Salesforce documentation on customizing inventory validation in the checkout process would detail the expected return types and values, ensuring that developers implement the inventory checks in a manner consistent with the platform's requirements.

Question: 197

What happens to all previous tax entries during tax implementation?

A. Modified with the new Tax calculation
B. They are deleted from the Cart
C. Saved prior to recalculation
D. Ignored with the recalculation

Answer: C

Explanation:

In general best practices for tax implementation in systems like Salesforce B2B Commerce, previous tax entries are usually preserved or saved before any recalculation is performed. This ensures that there is a record of the original tax calculations before any modifications, which can be crucial for auditing, historical data integrity, and in case the new tax calculation needs to be rolled back for any reason.

Question: 198

What should a developer's implementation code return if the External Prices are the same as Sales Prices for Products in the Cart?

A. sfde_checkout.IntegrationStatus. Status. SUCCESS
B. sfdc_checkout.IntegrationStatus. FAILED. Status
C. sfdc_checkout.IntegrationStatus.Status.FAILED
D. sfdc_checkout.IntegrationStatus.Success. STATUS

Answer: A

Explanation:

When implementing code for external pricing comparisons, if the external prices are the same as the sales prices for products in the cart, the implementation should ideally return a status indicating success (such as sfdc_checkout.IntegrationStatus.Status.SUCCESS). This indicates that the external pricing verification has passed and there are no discrepancies between the external and sales prices.

Question: 199

A developer is writing custom code to compare External Prices and Sales Prices for cart items. What should be returned if the External Prices are the same as Sales Prices for Products in the Cart? ~.

A. sfdc_checkout.IntegrationStatus.Status.SUCCESS
B. sfdc_checkout.IntegrationStatus.Status.SUCCEEDED
C. sfdc_checkout.IntegrationStatus.Status.ERROR
D. sfdc_checkout.IntegrationStatus.Success.FAILED

Answer: A

Explanation:

Similar to question 198, when custom code compares external prices with sales prices for cart items and finds them to be the same, the appropriate return value should indicate a successful comparison (such as sfdc_checkout.IntegrationStatus.Status.SUCCESS). This signifies that the external prices are aligned with the sales prices, and there are no conflicts.

Question: 200

What class must a developer implement to override Pricing during the checkout?

A. sfdc_commerce.CartPriceCalculations
B. sfdc_commerce.PriceCalculations
C. sfdc_checkout.PriceCalculations
D. sfdc_checkout.CartPriceCalculations

Answer: D

Explanation:

To override pricing during the checkout process in Salesforce B2B Commerce, a developer must implement a class specifically designed for this purpose, such as sfdc_checkout.CartPriceCalculations. This class would provide the necessary framework for custom pricing logic to be applied during checkout, ensuring that any custom pricing requirements are met.

Question: 201

What are two common and maintainable ways the content layout of a Lightning Web Component can be implemented?

A. Spreading layout styles across several separate components
B. Styling the :host pseudo-element (or other elements) via the CSS file
C. Using inline styles
D. Applying SLDS classes to internal elements

Answer: B, D

Explanation:

For maintainable and scalable Lightning Web Component development, it's recommended to style components by targeting the :host pseudo-element in the component's CSS file and by applying Salesforce Lightning Design System (SLDS) classes to internal elements. These practices ensure consistent styling that aligns with Salesforce's design standards and provides a clear separation of concerns between component structure and styling.

Question: 202

Which wire adapter can a developer use to retrieve metadata about a specific object?

A. getObject
B. getObjectMetadata
C. All of the above
D. getObjectinfo

Answer: D

Explanation:

To retrieve metadata about a specific object in a Lightning Web Component, a developer can use the getObjectInfo wire adapter. This adapter provides access to the metadata of a specified Salesforce object, including fields, record types, and layouts, which is essential for dynamic component rendering based on the object's schema.

Question: 203

Which two items are required for a developer to bring picklist values into a Lightning Web Component?

A. import { getPicklistvValues } from 'lightning/uiObjectinfoApi';
B. import { LightningElement, wire } from 'lwc';
C. import { wire } from 'lwc';
D. import { picklistValues } from 'lightning/uiObjectinfoApi';

Answer: A, B

Explanation:

To bring picklist values into a Lightning Web Component (LWC), a developer needs to import specific modules from the lwc and lightning/uiObjectInfoApi namespaces. The getPicklistValues function from the lightning/uiObjectInfoApi module is used to fetch the picklist values based on record type and field metadata. Additionally, importing { LightningElement, wire } from lwc is essential for defining the LWC class and using the @wire decorator to wire the getPicklistValues to a property or function. Salesforce documentation on LWC and utilizing the uiObjectInfoApi provides clear guidelines on how to implement this functionality.

Question: 204

Which three statements are accurate?

A. An Aura component can contain another Aura component
B. An Aura component can contain a Lightning Web Component
C. A Lightning Web Component can contain an Aura component
D. A Lightning Web Component cannot contain an Aura component

Answer: A, B, D

Explanation:

Salesforce documentation clarifies the interoperability between Aura and Lightning Web Components (LWCs). An Aura component can indeed contain another Aura component, as well as a LWC, allowing for a mix of component technologies in a single application. However, due to the architectural and design principles of LWCs, a LWC cannot contain an Aura component. This is because LWCs are designed to be lightweight and leverage web standards, which makes them not fully compatible with the older Aura component framework in terms of containment.

Question: 205

Which three decorators can be used in Lightning Web Components?

A. @api
B. @track
C. @wire
D. @class
E. @import

Answer: A, B, C

Explanation:

In Lightning Web Components, the decorators @api, @track, and @wire play crucial roles. The @api decorator is used to expose public properties and methods, making them accessible to other components. The @track decorator is used to mark private properties as reactive, so the UI updates when their values change. The @wire decorator is used to wire Apex methods or Salesforce data to the component. Salesforce documentation on LWC development extensively covers these decorators, explaining their usage and best practices.

Question: 206

Which three file extensions are allowed in a Lightning Web Component folder?

A. .js-meta.xml
B. .html
C. .Js
D. .gif
E. .jar

Answer: A, B, C

Explanation:

In a Lightning Web Component folder, the allowed file extensions include .js-meta.xml for the component's metadata,

.html for the component's markup, and .js for the component's JavaScript class. These files are essential for defining the structure, behavior, and metadata of a LWC. Salesforce LWC documentation provides detailed information on the structure of a LWC bundle and the purpose of each file type within it.

Question: 207

Which component can be used in other Salesforce Experience templates outside of B2B Commerce?

A. Quick Order
B. CMS Collection
C. Product Detail Data
D. Results Layout

Answer: B

Explanation:

In Salesforce Experience Cloud, components like CMS Collection and Results Layout are designed to be reusable across different Experience templates, not just limited to B2B Commerce. CMS Collection allows for the display of CMS content in a flexible and dynamic layout, while Results Layout can be used to present search or query results in a customizable format. Salesforce documentation on Experience Cloud components emphasizes the reusability and adaptability of these components across various templates and contexts.

Question: 208

What are two maintainable ways that Lightning Web Components can be made mobile ready? 33m 215

A. Create a Lightning app page and add the component to the mobile navigation
B. Import a third party JavaScript library
C. Install the mobile extensions plug-in for VS Code
D. Decorate templates with mobile-ready

Answer: A, D

Explanation:

To make Lightning Web Components mobile-ready, one maintainable approach is to create a Lightning app page and then add the component to the mobile navigation. This ensures that the component is accessible and optimized for mobile users within the Salesforce mobile app. Another approach is to design the component's templates with responsiveness in mind, using CSS and layout techniques that adapt to different screen sizes. Salesforce documentation on mobile-ready

development practices provides guidelines on creating responsive designs and optimizing components for mobile use.

Question: 209

A developer has just deployed a new Lightning Web Component to an authorized org. What should the developer do next to use the new component on apage?

A. Go to "Deploy LWC" in Setup.

B. Navigate 3 to the Page, Click on the "Custom Component Editor,' Click "Publish" on the new component in the list and adjust the positioning.

C. Create a Metadata API (MDAPI) conversion file with the Command Line interface (CLI) then go to the page and adjust the positioning.

D. Go to the page, edit it, and drag the new component onto the page.

Answer: D

Explanation:

After deploying a new Lightning Web Component (LWC) to an org, the typical next step to use the component on a page involves navigating to that page, entering the edit mode (usually through App Builder or a similar interface), and then dragging the new component from the components palette onto the desired location on the page. This process is consistent with Salesforce's modular design approach, allowing for easy integration and configuration of components within the Salesforce ecosystem.

Question: 210

A developer needs to deliver a solution for taxation that supports multiple countries and a complex set of jurisdictions. Which three steps should be considered as part of this process?

A. Contact the vendor who wrote the third party service for the most recent information
B. Implement the sfdc_checkout.CartTaxCalculations interface writing code from scratch
C. Implement the sfdc_checkout.CartTaxCalculations interface calling out to a third party tax service
D. Implement the sfdc_checkout.TaxCalculations interface calling out to a third party tax service
E. Look for packages or existing sample code on the AppExchange

Answer: CDE

Explanation:

For a solution that requires handling complex tax calculations across multiple countries and jurisdictions, it's efficient to leverage third-party tax services that specialize in this area. Implementing the sfdc_checkout.CartTaxCalculations or sfdc_checkout.TaxCalculations interface to integrate with such a service ensures that tax calculations are accurate and up-to-date. Additionally, exploring the Salesforce AppExchange for packages or sample code can provide ready-made solutions or frameworks that can expedite the development process.

Question: 211

A developer suspects that a defect exists in 30 lines of Apex code. How can the developer add debug statements, run the block of apex code in isolation and observe the immediate results?

A. Click the Check Out button in the storefront
B. Use the Execute Anonymous window
C. Activate Chrome dev tools and click the Check Out button in the Storefront
D. Use the Execute Immediate window

Answer: B

Explanation:

To debug a specific block of Apex code, the Execute Anonymous window in the Salesforce Developer Console or an Integrated Development Environment (IDE) like Visual Studio Code with the Salesforce Extension Pack can be used. This tool allows developers to run Apex code snippets in isolation and observe the results immediately, which is valuable for identifying and resolving defects within specific code blocks.

Question: 212

A developer suspects recent edits to a checkout flow have created a bug based on flow errors being emailed. Given the emails and some inputs known to trigger the issue, which two activities should the developer consider in their investigation?

A. Use the Org Browser tool in the IDE to download the flow XML and run a diff report
B. Look at the previous flow versions and compare them with the current one
C. Open the Flow, Select Debug, Provide the Inputs, Select Run
D. Open the Flow and select Attach to Live Session, Provide the Session Id, Select Attach

Answer: BC

Explanation:

To investigate a suspected bug in a checkout flow, comparing previous versions of the flow with the current one can help identify changes that might have introduced the bug. Additionally, Salesforce provides debugging capabilities within the Flow Builder, where a developer can select the debug option, provide input values known to trigger the issue, and execute the flow to observe its behavior, aiding in pinpointing the source of the problem.

Question: 213

Which two user permissions in addition to View Setup and Configuration are required to bulk create Product data translations via Data Loader?

A. Import Custom Objects
B. B2B Commerce Super User
C. Create and set up Experiences
D. Manage Translations

Answer: A, D

Explanation:

To bulk create product data translations via Data Loader, permissions beyond View Setup and Configuration are necessary. "Import Custom Objects" permission is required to import bulk data into Salesforce, including translations for custom objects. "Manage Translations" permission is essential for managing translation workbench settings and importing/exporting translation files, which is crucial for handling product data translations.

Question: 214

Which of these is a key pattern leveraged when building Lightning Web Components? 39m 36s

A. Composition

B. Inventory
C. Juggling
D. Normalization

Answer: A

Explanation:

Composition is a key pattern in building Lightning Web Components (LWCs). This approach involves creating small, reusable components that can be assembled to form more complex interfaces. This pattern promotes modularity, reusability, and maintainability in the development of web components, aligning with web standards and best practices.

Question: 215

What does the developer need to implement to override Shipping in Checkout? 38m 04s

A. sfdc_commerce.CartShippingCharges
B. sfdc_commerce ShippingCharges
C. sfdc_checkout.ShippingCharges
D. sfdc_checkout.CartShippingCharges

Answer: D

Explanation:

To override shipping charges during the checkout process, the developer needs to implement a specific interface, likely sfdc_checkout.CartShippingCharges. This interface would allow for custom logic to calculate and apply shipping charges based on the cart's contents and other criteria, ensuring flexibility and accuracy in handling shipping costs within the checkout flow.

Question: 216

How can a developer bring in a checkout flow step to another sequence order?

A. drag and drop checkout Screens in main checkout flow
B. drag and drop subflows in main checkout flow
C. Adjust next-state in previous subflow configuration
D. Reorder step in checkoutSteps.xml

Answer: C

Explanation:

In Salesforce B2B Commerce, to reorder a checkout flow step within a sequence, a developer must adjust the 'next-state' attribute in the configuration of the preceding subflow. This approach allows for dynamic control over the sequence of checkout steps without needing to modify the structure of the main checkout flow itself. The Salesforce B2B Commerce documentation outlines how checkout flows are constructed and how subflows can be managed and resequenced through configuration adjustments, providing a flexible and maintainable way to customize the checkout experience.

Question: 217

Which three considerations should a developer keep in mind when creating a tax provider?

A. What events to fire in the Lightning Web Component
B. Whether to use JSON or XML
C. Success criteria
D. Whether an AppExchange package already exists
E. How to handle errors

Answer: BCE

Explanation:

When creating a tax provider in Salesforce B2B Commerce, developers should consider the data format (JSON or XML) for interoperability with the tax service, define clear success criteria to ensure accurate tax calculations, and implement robust error handling to manage exceptions and failures gracefully. Salesforce B2B Commerce documentation emphasizes the importance of these considerations for integrating external services, ensuring reliability and consistency in tax calculations across different jurisdictions and scenarios.

Question: 218

Which technique is used by Lightning Web Components to provide areas of swappable, customizable content?

A. <slot> elements
B. JQuery templates
C. MutationObservers
D. CSS classes

Answer: A

Explanation:

Lightning Web Components (LWC) use <slot> elements to define areas within the component's template where content can be inserted dynamically. This feature, part of the Web Components standard, allows for the creation of customizable and reusable components. Salesforce developer documentation on LWC outlines the use of slots to build flexible component interfaces, enabling developers to design components that can adapt to various content structures and layouts.

Question: 219

What is one requirement to keep in mind when including additional JavaScript 1h files in a Lightning Web Component?

A. The files must be ES6 modules and must have names that are unique within the component's folder.
B. Only five of the files can be used with an import statement
C. All the files must be imported to a singleton.js file and the singleton.js file can be used with an import statement
D. Only one of the files can be used with an import statement

Answer: A

Explanation:

When including additional JavaScript files in a Lightning Web Component, it is required that these files are ECMAScript 6 (ES6) modules and have unique names within the component's folder. This ensures proper module resolution and avoids namespace conflicts. Salesforce LWC documentation provides guidelines on organizing component resources, including JavaScript modules, to ensure they are correctly recognized and utilized within the LWC framework.

Question: 220

Which tool is used to retrieve and manipulate Salesforce data in a Lightning Web Component?

A. Aura requests
B. Wire adapters
C. Proxy adapters
D. XHR requests

Answer: B

Explanation:

In Lightning Web Components, wire adapters are used to retrieve and manipulate Salesforce data. Wire adapters abstract the underlying data access mechanism, providing a declarative way to access Salesforce data and metadata. Salesforce documentation on LWC and the @wire decorator details how wire adapters can be used to connect components to Salesforce data sources, including standard and custom objects, facilitating reactive data binding and efficient data retrieval.

Question: 221

How can a developer make an integration available for selection?

A. Modify the StoreIntegrated Service to map to an Apex class id usingworkbench
B. Enter the integration class name and version in Store Administration
C. Create a RegisteredExternalService record using Workbench
D. The integration is available once it is uploaded

Answer: C

Explanation:

To make an integration available for selection in Salesforce B2B Commerce, a developer must create a RegisteredExternalService record. This can be accomplished using tools like Workbench, which allow for direct interaction with Salesforce's backend database. By registering the external service, the integration becomes selectable within the B2B Commerce setup, allowing it to be associated with specific stores or contexts. Salesforce documentation on integrating external services with B2B Commerce provides step-by-step instructions on registering these services to enable seamless integration and functionality within the platform.