Free Questions for B2B-Commerce-Developer by certsinside

Shared by Dickson on 22-07-2024

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

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

Options:

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 2

Question Type: MultipleChoice

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

Options:

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 3

Question Type: MultipleChoice

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? ~.

Options:

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 4

Question Type: MultipleChoice

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

Options:

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 5

Question Type: MultipleChoice

What happens to all previous tax entries during tax implementation?

Options:

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 6

Question Type: MultipleChoice

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?

Options:

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 7

Question Type: MultipleChoice

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

Options:

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 8

Question Type: MultipleChoice

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?

Options:

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 9

Question Type: MultipleChoice

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

Options:

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 10

Question Type: MultipleChoice

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

Options:

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 11

Question Type: MultipleChoice

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

Options:

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 12

Question Type: MultipleChoice

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

Options:

A- Validation rule

B- Apex class

C- Web service invocation

D- Lightning component load

E- Workflow invocations

Answer:

B, C, E

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.