Salesforce

AP-202

B2B Commerce for Developers Accredited Professional

For More Information – Visit link below:

https://www.examsempire.com/

Product Version

1. **Up to Date products, reliable and verified.**
2. **Questions and Answers in PDF Format.**

Latest Version: 6.0

Question: 1

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 B2B Commerce Developer Guide: Order Object, [B2B Commerce Developer Guide: Order Page], [Lightning Web Components Developer Guide: Call an Apex Method Imperatively]

Question: 2

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 {
   areDetailsVisible = false;
   handleChange(event) {
      this.areDetailsVisible = event.target.checked;
   }
}
```

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 Lightning Web Components Developer Guide: Conditional Rendering, Lightning Web Components Developer Guide: Template Syntax

Question: 3

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 Lightning Web Components Developer Guide: Lightning Data Service, Lightning Web Components Developer Guide: Work with Salesforce Data

Question: 4

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 Lightning Web Components Developer Guide: Debug Your Code, Lightning Web Components Developer Guide: JavaScript Limit Exceptions

Question: 5

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 self-closing, but rather have a closing tag (</target>). Salesforce Lightning Web Components Developer Guide: Configure Components for Lightning Communities, Lightning Web Components Developer Guide: Configure Components for Different Pages and Apps

Question: 6

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: C, D

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 B2B Commerce Developer Guide: Cart Item Object, B2B Commerce Developer Guide: Shipping Integration, B2B Commerce Developer Guide: Freight Integration

Question: 7

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 B2B Commerce Developer Guide: Tax Integration, B2B Commerce Developer Guide: Tax Calculation Flow

Question: 8

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 B2B Commerce Developer Guide: Payment Integration, B2B Commerce Developer Guide: Payment Gateway Adapter Interface, B2B Commerce Developer Guide: Payment Provider Gateway Object

Question: 9

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 B2B Commerce Developer Guide: Tax Integration, B2B Commerce Developer Guide: Tax Calculations Interface

Question: 10

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 B2B Commerce Developer Guide: Tax Integration, B2B Commerce Developer Guide: Tax Calculation Flow

Thank You for Trying Our Product

Special 16 USD Discount Coupon: NSZUBG3X

Email: support@examsempire.com

Check our Customer Testimonials and ratings available on every product page.

Visit our website.

https://examsempire.com/