DUMPSARENA

Adobe Commerce Architect Master

Adobe AD0-E722

Version Demo

Total Demo Questions: 5

Total Premium Questions: 52

Buy Premium PDF

https://dumpsarena.co

sales@dumpsarena.co

sales@dumpsarena.co
dumpsarena.co

Topic Break Down

Topic	No. of Questions
Topic 1, Architecture Design	12
Topic 2, Integration	6
Topic 3, Cloud Commerce	5
Topic 4, Customizations	28
Topic 5, Security	1
Total	52

QUESTION NO: 1

An Adobe Commerce Architect is asked by a merchant using B2B features to help with a configuration issue.

The Architect creates a test Company Account and wants to create Approval Rules for orders. The Approval Rules tab does not appear in the Company section in the Customer Account Menu when the Architect logs in using the Company Administrator account.

Which two steps must be taken to fix this issue? (Choose two.)

A. Set 'Enable B2B Quote' in the B2B Admin to TRUE

B. Merchant needs to log out of frontend and then log back in to load new permissions

C. Set 'Enable Purchase Orders' in the B2B Admin to TRUE

D. Set 'Enable Purchase Orders' on the Company Record to TRUE

E. Make sure that the 'Purchase Order' payment method is active

ANSWER: C E

Explanation:

Approval Rules are a B2B purchase-order capability in Adobe Commerce. To expose the Approval Rules area to a company administrator, the global B2B configuration must enable purchase orders by setting *Enable Purchase Orders* to *Yes*. This activates the purchase-order workflow and its associated company-account functionality, including the approval-rule management interface used to define conditions and approvers for orders that require authorization.

The *Purchase Order* payment method must also be enabled. Adobe's purchase-order documentation specifies that purchase orders must be enabled in the B2B features configuration and that the Purchase Order payment method is configured under payment methods. With both settings active, the company administrator can access the approval-rule features in the storefront company account and create rules appropriate to the company's ordering policy. These are store configuration prerequisites rather than permissions that can be enabled directly on an individual company record.

For the required B2B feature setting and storefront approval-rule workflow, see Adobe Commerce: Approval rules. For configuration of the required payment method, see Adobe Commerce: Purchase Order payment method.

QUESTION NO: 2

An Adobe Commerce Architect is reviewing API-functional test code. Some tests send errors to indicate that the customer address does not exist. The test codes show the following:

Which step should the Architect take to fix the test errors?

A. Change the annotation to use `@magentoApiDataFixture Magento/Customer/_files/customer_one_address.php` or `@magentoDataFixture Magento/Customer/_files/customer_one_address.php`.

B. Set the annotation to use `@magentoPersistDataFixture Magento/Customer/_files/customer_one_address.php` instead of `@magentoDataFixture Magento/Customer/_files/customer_one_address.php`.

C. Update the annotation to specify `address_id` in `@magentoDataFixture Magento/Customer/_files/customer_one_address.php` with `{"address_id":"$address.id$"}`.

ANSWER: B

Explanation:

Set the annotation to use `@magentoPersistDataFixture Magento/Customer/_files/customer_one_address.php` instead of `@magentoDataFixture`

`Magento/Customer/_files/customer_one_address.php`. API-functional tests can make application requests in a context separate from the test setup. The customer address fixture must therefore remain available when the API request executes. A regular `@magentoDataFixture` is designed for isolated test data handling and is rolled back as part of the test lifecycle. Consequently, data created through that fixture may not be present when a later API operation attempts to load the address, producing the reported “customer address does not exist” error.

The persistent-fixture annotation is intended for cases where fixture data must survive the normal rollback boundary and be available to subsequent operations or tests. Applying it to the customer-address fixture ensures that the address record is committed and can be resolved by the API-functional test code. The fixture itself remains the appropriate customer-address fixture; the required correction is its persistence behavior. Adobe Commerce documents fixture annotations and their lifecycle behavior in the Data fixture annotation documentation, and provides API testing guidance in the API-functional testing guide.

QUESTION NO: 3

An Adobe Commerce Architect is working on a scanner that will pull prices from multiple external product feeds. The Architect has a list of vendors and decides to create new config file marketplace.feeds.xml.

Which three steps can the Architect take to ensure validation of the configuration files with unique validation rules for the individual and merged files? (Choose three.)

A. Implement validation rules in the Converter class for the Config Reader

B. Create validation rules in marketplace.schema.xsd.

C. Provide schema to validate a merged file.

D. Add the Uniform Resource Name to the XSD file in the config XML file.

E. Provide schema to validate an individual file.

F. Create a class that implements \Magento\Framework\Config\Datainterface.

ANSWER: B C E

Explanation:

Creating validation rules in `marketplace.schema.xsd`, providing a schema to validate a merged file, and providing a schema to validate an individual file together implement Adobe Commerce’s two-stage configuration-validation pattern. The XSD defines the allowed XML structure for the custom configuration: valid elements, attributes, types, ordering, and constraints. Individual-file validation applies that definition when each module’s `marketplace.feeds.xml` is read, so every module contribution conforms to the expected format before it participates in configuration merging.

Merged-file validation is separately valuable because the effective configuration is the combined result of all discovered module configuration files. Its schema can enforce requirements that only make sense after merging, such as the resulting root structure, required final nodes, and valid aggregate relationships. Adobe Commerce config readers support distinct schema locations for per-file and merged validation, allowing an implementation to use rules appropriate to each phase. This helps identify invalid feed declarations early and ensures that the configuration consumed by the scanner is structurally reliable. Adobe’s XML configuration guidance describes using XSD schemas and separate validation for configuration loading and merged configuration: Adobe Commerce XML configuration documentation. The framework implementation is available in the Magento Framework Config source.

QUESTION NO: 4

An Adobe Commerce Architect needs to customize the workflow of a monthly installments payment extension. The extension is from a partner who is contracted with the default website Payment Service Provider (PSP), which has its own legacy extension (a module using deprecated payment method).

The installment payment partner manages only initializing a payment, and then hands the capture to be executed by the PSP Once the amount is successfully captured, the PSP notifies the website through a webhook. The goal of the webhook is only to create an "invoice" and save the "capture information" to be used later for refund requests through the PSP itself.

The Architect needs the most simple solution to capture the requested behavior.

Which solution should the Architect implement?

A. Add a plugin before the $invoice->capture() and change Its input to prevent the call of the $Payment->capture()

B. Change the can_capture attribute for the payment method under config.xml to be 0

C. Declare a capture Command with type Magento\Payment\Gateway\Command\NullCommand for the payment method CommandPool in di.xml

ANSWER: C

Explanation:

Declare a capture Command with type Magento\Payment\Gateway\Command\NullCommand for the payment method CommandPool in di.xml is the most appropriate implementation. Adobe Commerce payment integrations that use the payment gateway framework route operations such as capture through commands registered in the payment method's command pool. `NullCommand` implements the command contract but deliberately performs no gateway operation. Therefore, when invoice processing invokes the payment method's capture operation, Commerce can complete its local invoice and payment-state workflow without submitting a second capture request to the installments partner or to the PSP.

This matches the required division of responsibilities: the PSP performs the actual remote capture, then its webhook reaches Commerce after capture succeeds. The webhook integration can create the invoice and persist the PSP's capture or transaction data, which is needed to correlate later PSP refund requests. Using a no-operation capture command is also narrowly scoped to this payment method and follows the gateway framework's dependency-injection configuration model, making it a simple and maintainable adaptation of the payment flow. Adobe documents command pools and gateway commands at Adobe Commerce Payment Gateway commands; the framework's no-operation command implementation is available in the Magento Open Source NullCommand source.

QUESTION NO: 5

An Adobe Commerce Architect is planning to create a new action that will add gift registry items to the customer's quote. What should the Architect do to guarantee that private content blocks are updated?

A. Mark the controller by setting no-cache HTTP headers

B. Invalidate the status of gift registry indexers

C. Specify a new action in a sections.xml configuration file

ANSWER: C

Explanation:

Specify a new action in a sections.xml configuration file. Adobe Commerce uses customer-data sections to manage private content, such as customer-specific UI data that is loaded or refreshed in the browser rather than served from the full-page cache. The `sections.xml` configuration maps controller actions to the section data that must be invalidated and reloaded after those actions execute. For a custom action that adds gift registry items to a customer's quote, the module should declare the action's full route identifier under `<action name="...">` and list the applicable sections, typically including sections that expose quote state such as `cart`. When the action is performed, Commerce recognizes the configured action and causes the browser's customer-data mechanism to refresh the affected private-content sections. This ensures that customer-specific elements, including cart-related content, reflect the quote change without requiring the entire page to bypass caching. The action name must match the custom controller route, controller, and action in lowercase form. Adobe's private-content documentation describes this action-to-section mapping and the browser-side reload behavior, while the configuration reference provides the required `sections.xml` structure. See Adobe Commerce private content documentation and private-content configuration.