DUMPSBOSS.

Adobe Commerce Developer Expert

Adobe AD0-E709

Version Demo

Total Demo Questions: 8

Total Premium Questions: 83

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Architecture	42
Topic 2, EAV/Database	11
Topic 3, Layout/UI Customization	10
Topic 4, Checkout/Cart	9
Topic 5, Catalog	4
Topic 6, Sales	7
Total	83

QUESTION NO: 1

When paying by Bank Transfer, there is a requirement to send an email to customer service with payment details, after the order is placed successfully. Which two events can be used to send an email during the order placement process? (Choose two.)

A. sales_order_payment_pay

B. sales_model_service_auote_submit_before

C. sales_model_service_quote_submit_success

D. sales_order_place_after

ANSWER: C D

Explanation:

`sales_order_place_after` is dispatched by the sales order model immediately after its `place()` operation completes. An observer on this event has access to the order object, including its payment information, so it can identify the Bank Transfer payment method and initiate the required customer-service notification as part of order placement. `sales_model_service_quote_submit_success` is dispatched by the quote-submission service after the quote has been successfully converted and submitted as an order. It supplies both the resulting order and the originating quote, making it especially appropriate when the notification must be based on a successfully completed checkout submission and needs details from either object. In an implementation, the observer should check the order payment method before sending the message and should use an idempotent approach if notifications must not be duplicated during retries. These events are part of the sales and quote submission lifecycle exposed for observer-based customization. Adobe's event and observer guidance is available in the Adobe Commerce events and observers documentation; the corresponding dispatch points can also be reviewed in the Magento Sales Order source.

QUESTION NO: 2

An Adobe Commerce developer creates a new website using a data patch. Each website will have unique pricing by website. The developer does not have visibility into the production and staging environments so they do not know what the configuration currently is.

How would they ensure the configuration is deployed and consistent across all environments?

A)

Create a custom module and override the value in config.xml:

```
<?xml version="1.0" encoding="UTF-8" ?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchem
aLocation="urn:magento:module:Magento_Store:etc/config.xsd">
    <default>
        <catalog>
            <price>
                <scope>1</scope>
            </price>
        </catalog>
    </default>
</config>
```

B)

Run the CLI command below and commit the changes to the repository;

```
bin/magento config:set catalog/price/scope 1 --lock-config
```

C)

Run the CLI command below and commit the changes to the repository:

```
bin/magento config:set catalog/price/scope 1
```

A. Create a custom module and override the value in `config.xml`.

B. Run `bin/magento config:set catalog/price/scope 1 --lock-config` and commit the changes to the repository.

C. Run `bin/magento config:set catalog/price/scope 1 --lock-env` and commit the changes to the repository.

ANSWER: B

Explanation:

Run `bin/magento config:set catalog/price/scope 1 --lock-config` and commit the resulting change to the repository. The `catalog/price/scope` setting controls the Catalog Price Scope. A value of `1` sets the scope to Website, enabling each website to maintain its own product prices. This is the required global configuration before website-specific prices can be managed.

The `--lock-config` flag is essential because it writes the setting to the shared deployment configuration file, `app/etc/config.php`. That file is intended to be version controlled and deployed with the application code. Consequently, the same locked value is supplied to staging and production regardless of the pre-existing Admin configuration in either environment. Adobe Commerce imports deployed configuration during the configuration deployment process, making the repository version authoritative and preventing configuration drift between environments. This approach is particularly appropriate when a developer cannot inspect or manually align remote environment settings.

Adobe documents configuration locking and the distinction between shared configuration and environment-specific configuration in its configuration management CLI reference. The meaning and use of the Website Catalog Price Scope are documented in the Adobe Commerce Catalog Price Scope guide.

QUESTION NO: 3

An integration named Marketing is created on the Adobe Commerce instance. The integration has access on Magento_Customer::customer resources and the access token is xxxxxx .

How would the rest API be called to search the customers?

A)

Passing integration name and access token as http auth credentials:

```
curl -X GET https://Marketing:XXXXXX@magentourl/rest/V1/customers/search?searchCriteria...
```

B)

Using integration name as username and access token as password, get the admin token (YYYYYY) via:

```
curl -X POST https://magentourl/rest/V1/integration/admin/token -d '{"username":"Marketing", "password":"XXXXXX"}' -H 'Content-Type:application/json'
```

Use the admin token as Bearer `curl -X GET https://magentourl/rest/V1/customers/search?searchCriteria... -H 'Authorization: Bearer YYYYYY'`

C)

Using the integration access token as Bearer

```
curl -X GET https://magentourl/rest/V1/customers/search?searchCriteria... -H 'Authorization: Bearer XXXXXX'
```

A. Option A

B. Option B

C. Option C

ANSWER: C

Explanation:

Using the integration access token as Bearer is correct because an Adobe Commerce integration receives an OAuth access token after the integration authorization process. That token is supplied directly on subsequent REST requests in the HTTP `Authorization` header, in the form `Authorization: Bearer xxxxxx`. For this case, the request is made to the customer-search REST service, typically `GET /rest/<store_view_code>/V1/customers/search?searchCriteria=...` (or the equivalent endpoint without a store-view code where applicable). The integration's assigned ACL permission, `Magento_Customer::customer`, is evaluated when Adobe Commerce processes the request, allowing the token to access customer resources within that authorization scope.

An integration token is already the credential intended for API authentication; it does not need to be exchanged for a separate administrator token. The Bearer header also keeps authentication separate from query parameters and the request body, which is the standard REST authentication pattern documented for Adobe Commerce integrations. Adobe's REST authentication guidance describes using the integration access token as a Bearer token: Adobe Commerce REST authentication documentation. The customer API reference identifies the customer search operation and its search-criteria-based request structure: Adobe Commerce Customer REST API documentation.

QUESTION NO: 4

An Adobe Commerce developer is implementing a custom shipping carrier. The carrier should be displayed only when it is enabled and should return no shipping method when the destination country is not allowed by its configuration.

Which two actions should the developer take? (Choose two.)

A. Return `false` from `collectRates()` when the carrier is inactive.

B. Check the destination country against the configured allowed countries and return `false` when it is not allowed.

C. Hide the carrier with JavaScript after checkout rates have been returned.

D. Set the quote shipping method directly in an observer before totals collection.

ANSWER: A B

Explanation:

The carrier model should implement `collectRates(RateRequest $request)` and return `false` when the carrier is inactive. The shipping-rate collection process uses this method to determine whether the carrier supplies a rate result for the current quote and shipping address. Returning `false` prevents the carrier from contributing methods when it is disabled.

The implementation should also use the configured allowed-country setting to validate the request destination and return `false` when the destination is not permitted. This keeps country eligibility in the carrier's rate-collection logic and ensures unsupported destinations do not receive a selectable shipping method. Adobe's custom shipping carrier tutorial describes implementing rate collection and carrier configuration. The carrier contract can be reviewed in the CarrierInterface source.

QUESTION NO: 5

An Adobe Commerce developer has just finished creating a new custom entity, a block that extends \Mangento\Framework\View|Element\abstractBlock that lists all of the existing entities and a controller with the appropriate handle to display the block.

The developer now wants to implement cache on the block so that when one of the custom entities is saved, the cache of the block is automatically invalidated.

According to best practices- what are the two steps to accomplish this? (Choose two.)

A. 1. Override AbstractBlock: :getCecheKeyInfo and return an array containing the ids of all displayed entities.2. Override AbstractBlock: :getCacheTags and return an array containing, for all displayed entities, the value returned by the getcachetag method of the model

B. 1. Create an around plugin on the save( ) method of the model of the entity.2. Use the cleanCacheByTags( ) method Of \MagentoVFramework\App\Cache\FlushCaehebyTae with a single argument containing the concatenation of a chosen key and the td of the entity.

C. 1. Implement \Magento\Framework\DataObject\IdentityInterface on the block that lists the entities.2. Implement the getIdentities() method on the block to return an array containing the identities returned by all displayed entity models.

D. 1. Implement \Magento\Framework\DataObject\IdentityInterface on the model of the entity.2. Implement the getIdentities() method to return the entity cache tag, including the entity ID.

ANSWER: C D

Explanation:

The correct approach is to make both the entity model and the cached listing block identity-aware. The custom entity model implements `\Magento\Framework\DataObject\IdentityInterface` and returns a stable cache identity from `getIdentities()`, normally using the entity cache-tag prefix combined with its ID. Adobe Commerce uses these identities when the model is saved to clean cache entries carrying the corresponding tags.

The listing block also implements `IdentityInterface`. Its `getIdentities()` method aggregates the identities returned by every entity represented in the block. When block caching is enabled, these identities become cache tags for the rendered block output. Consequently, saving one of those entities triggers tag-based cache cleaning and invalidates the cached listing automatically. This keeps invalidation coupled to Commerce's standard cache-tag mechanism rather than requiring custom save plugins or manual cache-cleaning calls.

The identity contract is defined in the Adobe Commerce framework at Magento Framework IdentityInterface. Adobe's cache guidance also describes using cache tags and identities for invalidating cached content: Cache partial page content.

QUESTION NO: 6

An Adobe Commerce developer creates a new website using a data patch. Each website will have unique pricing by website. The developer does not have visibility into the production and staging environments so they do not know what the configuration currently is.

How would they ensure the configuration is deployed and consistent across all environments?

A. Create a custom module and override the value in config.xml:

B. Run the CLI command below and commit the changes to the repository;

C. Run the CLI command below and commit the changes to the repository:

ANSWER: B

Explanation:

Run the CLI command below and commit the changes to the repository; is correct because the `app:config:dump scopes` command exports the website, store, and store-view scope structure into the shared deployment configuration. This is particularly important when a data patch creates a website and catalog prices are intended to operate at website scope: the generated scope definitions must be version controlled so every environment receives the same website hierarchy and identifiers during deployment. The resulting configuration is saved in `app/etc/config.php`, which should be committed to source control along with the module and data patch. During deployment, Adobe Commerce imports the shared configuration, allowing staging and production to converge on the repository-defined state rather than depending on a developer manually inspecting or reproducing each environment's existing configuration. The scopes-specific dump is the appropriate export when the deployment concern includes websites and their related scopes; a general configuration dump alone does not provide the same targeted handling of scope definitions. Adobe documents configuration management and the use of configuration dump commands for deploying shared settings in its Configuration management guide and documents the configuration management CLI commands.

QUESTION NO: 7

An Adobe Commerce Developer has created a new custom block extending \Magento\Framework\view\Element\AbstractBlock and has set the cache_lifetime data property for the block so that the output gets cached.

The block is inserted into the sidebar, and displays differing content depending on which currency is being used. The developer finds that the block is displaying the same content for all currencies, depending on which currency is viewed first after the cache has been flushed.

How would the developer resolve this?

A. Implement the \Magento\Framework\DataObject\Identityinterface class, as well as a getIdentities() method, returning the current currency code.

B. Override the getCacheKeyinfo() function adding the current currency code to the returned array.

C. In the constructor, add the current currency code as a cache tag using $thismetaDataCcache_tags'', CURRENCY_CODE]) .

ANSWER: B

Explanation:

Override the getCacheKeyinfo() function adding the current currency code to the returned array. is correct because a cacheable block's rendered HTML is stored and retrieved using its cache key. The key must include every piece of context that can change the block's output. Since the custom block renders different content for each selected currency, the current currency code is required key context. Including it in the array returned by `getCacheKeyInfo()` causes Adobe Commerce to generate a distinct cache entry for each currency. As a result, a visitor using one currency receives the HTML rendered for that currency rather than the entry initially populated under another currency.

The implementation should retain the parent key information and append a stable currency identifier, typically the current store's currency code. For example, the method can obtain the code from the store manager and add it to the array returned by `parent::getCacheKeyInfo()`. Adobe Commerce's `AbstractBlock` uses this method when composing the block cache key, making it the appropriate extension point for output variations such as currency-specific rendering. Adobe's cache guidance emphasizes that cache identities and keys must account for the data and context represented by cached content. See the Adobe Commerce caching guide and the AbstractBlock source implementation.

QUESTION NO: 8

An Adobe Commerce developer is working on an Acme_Exceptions module which is supposed to overwrite logic inside some of Magento native exceptions such as \Magento\Frame-work\Exception\NoSuchEntityException or \Mogento\Framwork\GraphQ1\Exception\GrephQlInputException , The module is open-source and will be available on packagist.org.

The build of the codebase of projects, including the module, will sometimes take place in docker containers with full access to filesystem. but then it is deployed to a read-only filesystem.

Which two approaches would the developer use to overwrite logic in those exceptions? (Choose two.)

A. 1. Create a version of those exceptions inside the module using the original namespaces and classes, e.g. \Magento\Framework\Exception\NoSuchEntityException.
2. Use Composer's `extra.patches` configuration to apply a patch that replaces the original files in locations such as `vendor/magento/framework/Exception/`.

B. 1. Create a version of those exceptions inside the module using new namespace, e.g. \Acme\Exceptions\Exception\NoSuchEntityException .

C. 1. Create a version of those exceptions inside the module using the original namespaces and classes, e.g. \Magento\Framework\Exception\NoSuchEntityException.

2. Use Composer's `scripts.post-install-cmd` and `scripts.post-update-cmd` entries to copy those exceptions to their original destinations, such as `vendor/magento/framework/Exception/`, replacing the original files.

D. 1. Create a version of Those exceptions inside the module using the original namespaces and classes, e.g. \Magento\Framework\Exception\NoSuchEntityException .
2. Create a data patch copying those exceptions to their original destinations like vendor/magento/ framework /except Ion/ to replace original files.

ANSWER: A C

Explanation:

The Composer `extra.patches` approach is appropriate because it modifies the installed dependency source while the project is being built, before the immutable deployment artifact is created. A patch can replace the implementation of a Magento framework exception at its installed path, so runtime code resolves the altered class from the normal Magento namespace. In practice, the consuming Commerce project should declare and apply the patch as part of its reproducible build configuration; the patch file can be maintained by the Acme package or project repository. Composer Patches documents patch declaration through the `extra.patches` configuration: Composer Patches documentation.

Using Composer `post-install-cmd` and `post-update-cmd` scripts is also a build-time solution. Those scripts can copy the maintained replacement exception files into the corresponding installed `vendor/magento/framework` locations after Composer has installed or updated dependencies. This must occur in the writable build environment, and the resulting vendor tree must then be packaged and deployed as the read-only artifact. Composer documents these command events as scripts that run after installation and update operations: Composer scripts documentation. Both approaches therefore ensure that the replacement is already present before the application runs on a read-only filesystem.