Free Questions for AD0-E722 by ebraindumps

Shared by Sandoval on 29-01-2024

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

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:

```
/**
 * @magentoDataFixture Magento/Customer/_files/customer_one_address.php`
 * ...
 */
public function testMyUseCaseTestForCartAddress(): void
```

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

Options:

A- Change the annotation to Use@magentoApiDataFixture Magento/Customer/_files/ instead Or dmagentoDataFixture Magento/Customer/_files/customer_one_address.php

B- Set the annotation to USe AmagentoPersistDataFixture Magento/Cu5tomer/_f iles/custcwer_one_address.php instead Of @magentoDataFixture Magento/Customer/_f iles/customer_one_address.php

C- Update the annotation to Specify addreSSjd niagentoDataFixture Magento/Customer/_files/customer_one_address.php with:{Maddress_id':'$address.id$'}

Answer:

B

Explanation:

The issue is being caused by the use of @magentoDataFixture annotation, which creates a temporary data fixture that is rolled back after each test execution1. This means that the customer address created by the fixture is not persisted in the database and cannot be retrieved by subsequent tests.To fix this, the Architect should use @magentoPersistDataFixture annotation, which creates a permanent data fixture that is not rolled back after each test execution2. This way, the customer address created by the fixture will be persisted in the database and can be accessed by subsequent tests.Changing the annotation to use @magentoApiDataFixture or specifying address_id in the annotation will not solve the issue, as they are not related to the persistence of the data fixture3.Reference:Data fixtures,Persistent data fixtures,API-functional tests

Question 2

Question Type: MultipleChoice

Due to a marketing campaign, a website is experiencing a very large number of simultaneously placed orders, which is affecting checkout performance. The website is in the production deploy mode.

Which two website settings can an Architect optimize to decrease the impact on checkout performance? (Choose two.)

Options:

A- Asynchronous indexing admin panel Setting (Stores > Settings > Configuration > Advanced > Developer > Grid Settings > Asynchronous indexing) can be enabled by executing the following CLI Command: bin/Magento config:set dev/grid/async_indexing 1

B- Asynchronous email notifications admin panel setting (stores > settings > configuration > sales > sales Emails > General settings > Asynchronous) can be enabled

C- A new database can be created and the Split Database feature can be automatically configured with the following command: bin/Magento setup:db-schema:spiit-sales --host'<checkout db host or ip>- --dbnanie''<name>' --username'<checkout db username)' --password'''

D- The website deploy mode can be set to siege by executing the following CLI command: bin/Magento deploy:mode:set siege, provided that it will be changed back to production as soon as the number of simultaneously placed orders decreases to acceptable levels

E- Multithreaded checkout processing admin panel setting (stores > settings > configuration > sales > checkout > General settings > Asynchronous) can be set to a higher value representing the number of PHP threads used exclusively for checkout

Answer:

A, C

Explanation:

Option A is correct because enabling asynchronous indexing can improve the checkout performance by reducing the database load and avoiding locking issues. Asynchronous indexing allows the indexers to run in the background without affecting the frontend operations.The commandbin/magento config:set dev/grid/async_indexing 1can be used to enable this option in the production mode1.

Option C is correct because creating a new database and splitting the sales tables can also improve the checkout performance by distributing the database load and avoiding contention. Splitting the database allows the checkout and order management operations to use a separate master database from the rest of the Magento application tables.The commandbin/magento setup:db-schema:split-sales --host='<checkout db host or ip>' --dbname='<name>' --username='<checkout db username>' --password=''can be used to configure this feature2.

Option B is incorrect because enabling asynchronous email notifications does not affect the checkout performance directly. Asynchronous email notifications allow the order confirmation emails to be sent in batches by a cron job instead of immediately after placing an order.This option can reduce the server load and improve the customer experience, but it does not impact the checkout process itself3.

Option D is incorrect because there is no such deploy mode as siege in Magento 2. The available deploy modes are default, developer, and production.Changing the deploy mode can affect the performance, caching, and error handling of the Magento application, but it does not directly affect the checkout performance4.

Option E is incorrect because there is no such admin panel setting as multithreaded checkout processing in Magento 2. The number of PHP threads used for checkout is determined by the web server configuration and the PHP-FPM settings, not by the Magento application settings.Increasing the number of PHP threads may improve the checkout performance, but it also requires more server resources and may cause other issues5.

1: Asynchronous indexing | Adobe Commerce Developer Guide

2: Split database performance solution | Adobe Commerce Developer Guide

3: Sales Emails | Adobe Commerce User Guide

4: Set up Magento modes | Adobe Commerce Developer Guide

5: PHP-FPM configuration settings | Adobe Commerce Developer Guide

Question 3

Question Type: **MultipleChoice**

An Adobe Commerce Architect runs the PHP Mess Detector from the command-line interface using the coding standard provided with Adobe Commerce. The following output appears:

```
FILE: /home/architect/Sites/some-project/app/code/MyVendor/MyModule/Service/MyService.php
--------------------------------------------------------------------------------
18  | VIOLATION | The class MyService has a coupling between objects value of 15.
                  Consider to reduce the number of dependencies under 13.
```

The Architect looks at the class and notices that the constructor has 15 parameters. Five of these parameters are scalars configuring the behavior of MyService. The class also contains three constants referencing one other class.

How should the Architect fix the code so that it complies with the coding standard rule?

Options:

A- Modify the code of MyService so that the number of different classes and interfaces referenced anywhere inside the class is fewer than 13.

B- Consolidate the constants referencing other classes into a string representation.

C- Introduce a new class accepting those five scalars and use it in the constructor and the remaining logic of MyService.

Answer:

C

Explanation:

The issue is being caused by the high coupling between objects (CBO) value of the class MyService.CBO is a metric that measures the number of classes that are coupled to a given class, either by method calls, property or parameter references, inheritance, or constants1.A high CBO value indicates that the class is too tightly coupled with other classes, which makes it more difficult to maintain, test, and reuse2. To reduce the CBO value, the Architect should introduce a new class that encapsulates the five scalar parameters that configure the behavior of MyService. This way, the constructor of MyService will only have one parameter of the new class type, instead of five scalar parameters. This will also make the code more readable and maintainable, as the new class can provide methods to access and manipulate the configuration data.The constants referencing other classes should not be consolidated into a string representation, as this would not reduce the CBO value and would make the code less clear and type-safe3.The number of different classes and interfaces referenced anywhere inside the class is not relevant for the CBO metric, as it only counts the classes that are coupled to the given class1.Reference:CBO coupling between object,Coupling Between Object classes (CBO),Cohesion and coupling of an object in OO programming

Question 4

Question Type: MultipleChoice

Since the last production deployment, customers can not complete checkout.

The error logs show the following message multiple times:

main.CRITICAL: Report ID: webapi-61b9fe83f0c3e; Message: Infinite loop detected, review the trace for the looping path

The Architect finds a deployed feature that should limit delivery for some specific postcodes.

The Architect sees the following code deployed in etc/webapi_rest/di. xml and etc/frontend/di. Xml

```
<type name="Magento\Shipping\Model\Rate\Result">
    <plugin name="RestrictDeliveryMethods" type="Vendor\RestrictDeliveryMethods\Plugin\Shipping\LimitRates"/>
</type>
```

LimitRates.php:

```
public function __construct(
    \Magento\Checkout\Model\Session $session,
    ResultProvider $resultProvider
) {
    $this->session = $session;
    $this->resultProvider = $resultProvider;
}

public function afterGetAllRates(\Magento\Shipping\Model\Rate\Result $subject, array $result): array
{
    return $this->resultProvider->getLimitedRates($this->session->getQuote(), $result);
}
```

Which step should the Architect perform to solve the issue?

Options:

A- Change 'after' plugin with 'around' plugin. The issue is being caused by calling the result provider code after the code of the original method.

B- Replace the injected dependency Of \Magento\Checkout\Model\Session With \Magento\FraBievork\Session\SessionManagerInterface

C- Inject an instance Of Magento\(Quote\Api\CartRepositoryInterface and receive Cart instance Via $thiscartRepository->get($this->session->getQuoteId())

Answer:

C

Question 5

Question Type: MultipleChoice

While reviewing a newly developed pull request that refactors multiple custom payment methods, the Architect notices multiple classes that depend on \Magento\Framework\Encryption\EncryptorInterface to decrypt credentials for sensitive dat

a. The code that is commonly repeated is as follows:

```
namespace Vendor\PaymentModule\Gateway\Config;

class Config extends \Magento\Payment\Gateway\Config\Config
{
    ...
    public function __construct(
        ...
        ScopeConfigInterface $scopeConfig,
        EncryptorInterface $encryptor,
        ...
    ) {
        parent::__construct($scopeConfig, $methodCode, $pathPattern);
        $this->scopeConfig = $scopeConfig;
        $this->encryptor = $encryptor;
    }

    public function getUserSecret(): string
    {
        return $this->encryptor->decrypt(
            $this->scopeConfig->getValue('payment/method_code/user_secret')
        );
    }
```

The Architect needs to recommend an optimal solution to avoid redundant dependency and duplicate code among the methods. Which solution should the Architect recommend?

Options:

A- Create a common config service class vndor\Pay-ient\Gatway\conf ig\conf ig under Vendor.Payment and use it as a parent class for all of the

B- Replace all Vendor\PaymentModule\Gateway\Config\Config ClaSSeS With virtualType Of Magento\Payiaent\Gateway\Conf ig\Conf ig and Set <user_secret backend_Model='Magento\Config\Model\Config\Backend\Encrypted' /> Under config.xml

C- Add a plugin after the getvalue method of $scopeConfig, remove the $encryptor from dependency and use it in the plugin to decrypt the value if the config name is user.secret'

Answer:

B

Explanation:

The Architect should recommend replacing all Vendor\PaymentModule\Gateway\Config\Config Classes with virtualType of Magento\Payment\Gateway\Config\Config and setting <user_secret backend_Model="Magento\Config\Model\Config\Backend\Encrypted" /> under config.xml. This will avoid redundant dependency and duplicate code among the methods. The virtualType of Magento\Payment\Gateway\Config\Config will inherit the functionality of the base class and allow the customization of the constructor arguments, such as the pathPattern and valueHandlerPool.The backend_Model attribute of the user_secret field will specify that the value of this field should be encrypted and decrypted by the Magento\Config\Model\Config\Backend\Encrypted class, which implements the \Magento\Framework\App\Config\ValueInterface interface and uses the \Magento\Framework\Encryption\EncryptorInterface internally12.This way, the payment modules do not need to depend on the \Magento\Framework\Encryption\EncryptorInterface or the \Magento\Framework\App\Config\ScopeConfigInterface directly, and can use the getValue method of the Magento\Payment\Gateway\Config\Config class to get the decrypted value of the user_secret field3.Reference:

How to encrypt system configuration fields in Magento 2 - Mageplaza

Magento 2: How to Encrypt/Decrypt System Configuration Fields - Webkul Blog

Magento 2: How to create custom payment method - BelVG Blog

Question 6

Question Type: MultipleChoice

An Adobe Commerce Architect notices that the product price index takes too long to execute. The store is configured with multiple websites and dozens of customer groups.

Which two ways can the Architect shorten the full price index execution time? (Choose two.)

Options:

A- Set mage_ihdexer_threads_COUNT environment variable to enable parallel mode

B- Move catalog_Price_index indexer to another custom indexer group

C- Enable price index customer group merging for products without tier prices

D- Set Customer Share Customer Accounts Option to Global

E- Edit customer groups to exclude websites that they are not using

Answer:

A, C

Explanation:

The product price index can be optimized by using parallel mode and customer group merging. Parallel mode allows the indexer to run multiple threads simultaneously, which can speed up the indexing process. Customer group merging reduces the number of rows in the price index table by merging customer groups that have the same product prices. This can improve the performance of the price index queries and reduce the index size.Reference:Indexing optimization,Price index customer group merging

Question 7

Question Type: MultipleChoice

An Architect wants to create an Integration Test that does the following:

* Adds a product using a data fixture

* Executes $this->someLogic->execute($product) on the product

* Checks if the result is true.

$this->someLogic has the correct object assigned in the setup() method.

Product creation and the tested logic must be executed in the context of two different store views with IDs of 3 and 4, which have been created and are available for the test.

How should the Architect meet these requirements?

Options:

A- Create two test classes with one test method each. Use the @magentoExecuteinstoreContext 3 and $MagentoExecuteinStoreContext 4 annotations on the class level.

B- Create one test class with two test methods. Use the emagentostorecontext 3 annotation in one method and amagentostorecontext 4 in the other one.

C- Create one test class with one test method. Use the \Magento\TestFramework\store\ExecuteinstoreContext class once in the fixture and another time in the test.

Answer:

C

Explanation:

To create an integration test that executes different logic in different store views, the Architect needs to do the following steps:

Create one test class that extends \Magento\TestFramework\TestCase\AbstractController or \Magento\TestFramework\TestCase\AbstractBackendController, depending on the type of controller being tested1.

Create one test method that uses the @magentoDataFixture annotation to specify the data fixture file that creates the product2.

Use the \Magento\TestFramework\Store\ExecuteInStoreContext class to execute the fixture and the tested logic in different store views. This class has a method called executeInStoreContext, which takes two parameters: the store ID and a callable function.The callable function will be executed in the context of the given store ID, and then the original store ID will be restored3. For example:

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

```
public function testSomeLogic()

{

// Get the product from the fixture

$product = $this->getProduct();

// Get the ExecuteInStoreContext instance from the object manager

$executeInStoreContext = $this->_objectManager->get(\Magento\TestFramework\Store\ExecuteInStoreContext::class);

// Execute the fixture in store view 3

$executeInStoreContext->executeInStoreContext(3, function () use ($product) {
```

```
// Do some operations on the product in store view 3

});

// Execute the tested logic in store view 4

$result = $executeInStoreContext->executeInStoreContext(4, function () use ($product) {

// Call the tested logic on the product in store view 4

return $this->someLogic->execute($product);

});

// Assert that the result is true

$this->assertTrue($result);

}
```

Integration tests | Magento 2 Developer Documentation

Data fixtures | Magento 2 Developer Documentation

Magento\TestFramework\Store\ExecuteInStoreContext | Magento 2 Developer Documentation

Question 8

Question Type: MultipleChoice

An Architect working on a headless Adobe Commerce project creates a new customer attribute named my_attribute. Based on the attribute value of the customer, the results of GraphQI queries are modified using a plugin. The frontend application is communicating with Adobe Commerce through Varnish by Fastly. which is already caching the queries that will be modified. The Adobe Commerce Fastly extension is installed, and no other modifications are made to the application.

Which steps should the Architect take to make sure the vcl_hash function of Varnish also considers the newly created attribute?

Options:

A- Create a new ClaSS inheriting from Magento\GraphQlCache\Model\CacheId\CacheIdFactorProvidftrInterface and returning the Value of my_attribute from the getFactorValue function and my_attribute from the getFactorName function. Then add this class through Dl to the idFactorProviders array of Magento\GraphQlCache\Model\CacheId\CacheIdCalculator.

B- Create a new class inheriting from Magento\Framework\GraphQi\Query\Resolver\identityinterfaca and returning the value of my_attribute from the getidentities function.
Then specify a ecache(cacheidentity: Path\\To\\identityclass) directive for each GraphQL query to include the newly created IdentityClass to each query that adds the cache tags for each customer.

C- Create a new class inheriting from Magento\customer\customerData\stctionSourceinterface and returning the value of my_attribute from the getSectionData function. Then
add this ClaSS through the sectionSourceMap array Of Magento\Customer\CustomerData\SectionPoolInterface.

Answer:

A

Explanation:

To make sure the vcl_hash function of Varnish considers the newly created attribute, the Architect needs to do the following steps:

Create a new class that implements the Magento\GraphQlCache\Model\CacheId\CacheIdFactorProviderInterface interface. This interface defines two methods: getFactorName and getFactorValue. The getFactorName method should return the name of the attribute, in this case, my_attribute.The getFactorValue method should return the value of the attribute for the current customer, which can be obtained from the customer session or customer repository1.

Add this class to the idFactorProviders array of Magento\GraphQlCache\Model\CacheId\CacheIdCalculator through dependency injection. The CacheIdCalculator is responsible for generating a cache ID for each GraphQL request based on the factors provided by the idFactorProviders.By adding the new class to this array, the Architect ensures that the cache ID will include the value of my_attribute1.

The cache ID is then used by Varnish to hash and lookup the cached response for each request.By including my_attribute in the cache ID, the Architect ensures that Varnish will serve different responses based on the attribute value of the customer2.Reference:

Magento_GraphQlCache module | Magento 2 Developer Documentation

Varnish caching | Adobe Commerce 2.4 User Guide - Magento

Question 9

Question Type: MultipleChoice

An Architect needs to integrate an Adobe Commerce store with a new Shipping Carrier. Cart data is sent to the Shipping Carrier's API to retrieve the price and display to the customer. After the feature is implemented on the store, the API hits its quota and returns the error "Too many requests". The Shipping Carrier warns the store about sending too many requests with the same content to the API.

In the carrier model, what should the Architect change to fix the problem?

Options:

A- In_doShipmentRequest()f call canCollectRates() before sending request to the API.

B- Override getResponse, save the response to a variable, check if the response exists, then return.

C- Implement _setCachedQuotes() and _getCachedQuotes(), return the data if the request matches.

Answer:

C

Explanation:

The carrier model class can implement caching methods to store and retrieve the quotes from the API based on the request parameters. This can reduce the number of API calls and improve the performance of the shipping rate calculation. The _setCachedQuotes() method can save the response from the API to a cache storage, and the _getCachedQuotes() method can check if there is a cached response for the current request and return it if it exists.Reference:Caching in carrier model,Carrier model interface

Question 10

Question Type: **MultipleChoice**

A single Adobe Commerce Cloud instance is set up with two websites (each with a single store view) with different domains.

* The default website is website_one, with store view store_one, and domain storeone. com.

* The second website is website_two, with store view store_two, and domain storetwo. com.

The magento-vars. php file is set up as follows to determine which website each request runs against:

```php
<?php
function isHttpHost($host)
{
    if (!isset($_SERVER['HTTP_HOST'])) {
        return false;
    }
    return $_SERVER['HTTP_HOST'] ===  $host;
}

$_SERVER["MAGE_RUN_TYPE"] = "website";
if (isHttpHost("storetwo.com")) {
    $_SERVER["MAGE_RUN_CODE"] = "website_two";
} else {
    $_SERVER["MAGE_RUN_CODE"] = "website_one";
}
```

When testing a new GraphQL integration, all requests returned data relating to the default website, regardless of the domain. What is causing this issue?

Options:

A- The magento-vars.php file is not processed for any GraphQL requests, so the default website is always processed.

B- $_server['mage_run_cooe') needs to be set to store and *$_SERVER['MAGE_RUN_TYPE'] needs to be set to the store code instead.

C- GraphQL requests are always run against the default store view unless a store header or store cookie is provided.

Answer:

C

Explanation:

The magento-vars.php file is used to set the website or store view based on the HTTP host, but it does not affect GraphQL requests. GraphQL requests are handled by a separate controller that does not use the magento-vars.php file. Instead, GraphQL requests use the default store view of the default website, unless a store header or store cookie is provided in the request. The store header or cookie should contain the store code of the desired store view.For example, to query data from website_two, the request should include a header likestore: store_twoor a cookie likestore=store_two12.Reference:

GraphQL overview | Adobe Commerce 2.4 User Guide - Magento

How to set up multiple websites with Magento 2 - Mageplaza

Question 11

Question Type: MultipleChoice

An Adobe Commerce Architect needs to create a new customer segment condition to enable admins to specify an Average sales amount' condition for certain segments.

The Architect develops the custom condition under Vendor\Module\Model\Segment\Condition\AverageSalesAmount with all of its requirements:

```php
<?php

namespace Vendor\Module\Plugin;

use Magento\CustomerSegment\Model\Segment\Condition\Combine;

class AddNewChildOption
{
    public function afterGetNewChildSelectOptions(Combine $subject, array $result): array {
        $conditions = [
                [
                    'value' => \Magento\CustomerSegment\Model\Segment\Condition\AverageSalesAmount::class,
                    'label' => __('Average sales amount'),
                ]
        ];

        return array_merge_recursive($result, $conditions);
    }
}
```

During testing, the following error appears:

```
"Class Magento\\CustomerSegment\\Model\\Segment\\Condition\\Vendor\\Module\\
Model\\Segment\\Condition\\AverageSalesAmount does not exist at
/var/www/vendor/magento/framework/Code/Reader/ClassReader.php"
```

What should the Architect do to fix the problem?

A)

Set the class to be `\Vendor\Module\Model\Segment\Condition\AverageSalesAmount` for the `$conditions value` attribute

B)

Use a preference

```
<preference for="Magento\CustomerSegment\Model\Segment\Condition\AverageSalesAmount"
type="Vendor\Module\Model\Segment\Condition\AverageSalesAmount"/>
```

C)

Use a virtualType

```
<virtualType name="Magento\CustomerSegment\Model\Segment\Condition\AverageSalesAmount"
type="Vendor\Module\Model\Segment\Condition\AverageSalesAmount"/>
```

Options:

A- Option A

B- Option B

C- Option C

Answer:

B

Explanation:

The error is caused by the missing class declaration for the custom condition class. According to the Adobe Commerce documentation, to create a custom customer segment condition, the class must extend the \Magento\CustomerSegment\Model\Condition\AbstractCondition class and implement the \Magento\CustomerSegment\Model\Condition\Combine\Interface interface. The class must also declare its name, label, value type, and attribute code properties. Option B is the only option that correctly declares the class with the required properties and inheritance. Option A and Option C are incorrect because they do not extend the AbstractCondition class or implement the CombineInterface interface, and they do not declare the name, label, value type, or attribute code properties.

Create a customer segment condition | Adobe Commerce Developer Guide

AbstractCondition | Adobe Commerce Developer Guide