Free Questions for AD0-E716 by actualtestdumps

Shared by Suarez on 24-05-2024

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

An Adobe Commerce developer is working on a custom gallery extension.

The module uses the Magento\catalog\Model\iinageUploader class for image uploading. The admin controller for custom image uploads is Vendor\CustomGallery\Controller\Adminhtml\Image\Upload.

The images need to be stored in different basePath and baseTmpPath than the default ones.

How can the default imageuploader class be extended and used without affecting the other modules that are already using it?

A)

1. Create a Virtual Type and configure the `basePath` and `baseTmpPath`.
2. Inject the virtual type `Vendor\CustomGallery\GalleryImageUpload` into admin controller:

```
<virtualType
  name="Vendor\CustomGallery\GalleryImageUpload"
  type="Magento\Catalog\Model\ImageUploader"
>
    <arguments>
        <argument name="baseTmpPath" xsi:type="string">customgallery/tmp/images</argument>
        <argument name="basePath" xsi:type="string">customgallery/images</argument>
    </arguments>
</virtualType>

<type name="Vendor\CustomGallery\Controller\Adminhtml\Image\Upload">
    <arguments>
        <argument name="imageUploader" xsi:type="object">
            Vendor\CustomGallery\GalleryImageUpload
        </argument>
    </arguments>
</type>
```

B)

1. Configure the `basePath` and `baseTmpPath` of `Magento\Catalog\Model\ImageUploader`.
2. Inject the type `Magento\Catalog\Model\ImageUploader` into admin controller:

```
<type name="Magento\Catalog\Model\ImageUploader">
    <arguments>
        <argument name="baseTmpPath" xsi:type="string">customgallery/tmp/images</argument>
        <argument name="basePath" xsi:type="string">customgallery/images</argument>
    </arguments>
</type>

<type name="Vendor\CustomGallery\Controller\Adminhtml\Image\Upload">
    <arguments>
        <argument name="imageUploader" xsi:type="object">
            Magento\Catalog\Model\ImageUploader
        </argument>
    </arguments>
</type>
```

C)

1. Create a Virtual Type and configure the `basePath` and `baseTmpPath`.
2. Add virtual type `Vendor\CustomGallery\GalleryImageUpload` as a preference for `Magento\Catalog\Model\ImageUploader:

```
<virtualType
  name="Vendor\CustomGallery\GalleryImageUpload"
  type="Magento\Catalog\Model\ImageUploader">
    <arguments>
        <argument name="baseTmpPath" xsi:type="string">customgallery/tmp/images</argument>
        <argument name="basePath" xsi:type="string">customgallery/images</argument>
    </arguments>
</virtualType>

<preference
    for="Magento\Catalog\Model\ImageUploader"
    type="Vendor\CustomGallery\GalleryImageUpload"
/>
```

Options:

A- Option A

B- Option B

C- Option C

Answer:

B

Explanation:

According to the ImageUploader component guide for Magento 2 developers, the ImageUploader UI component gives users the ability to upload images to the Magento Media Gallery. This component is a variation of the FileUploader component and uses the same configuration settings. The ImageUploader component uses the Magento\catalog\Model\iinageUploader class for image uploading, which has properties such as basePath and baseTmpPath that define where the images are stored. To extend the default imageuploader class and use it without affecting the other modules that are already using it, the developer needs to create a virtual type of this class in their module's di.xml file and specify different values for basePath and baseTmpPath. The developer also needs to inject their virtual type into their admin controller using the imageUploader argument. Therefore, option B is the correct answer, as it shows the correct di.xml and controller code to extend and use the imageuploader class. Verified Reference: https://devdocs.magento.com/guides/v2.3/ui_comp_guide/components/image-uploader/

Question 2

Question Type: MultipleChoice

An Adobe Commerce developer is asked to change the tracking level on a custom module for free downloading of pdf and images.

The module contains following models:

Download class has a parameter for tracking_level.

How will the developer configure the tracking_level parameter, in di.xml.to have a value of 4 for Download class and all classes that extend Download?

A)

Configure the parameter on a child class and add `parent` attribute as it will be propagated to siblings and parent.

```
<type
  name="Vendor\FreeDownload\Model\DownloadPdf"
  parent="Vendor\FreeDownload\Model\Download"
>
    <arguments>
        <argument name="tracking_level" xsi:type="integer">4</argument>
    </arguments>
</type>
```

B)

Configure the parameter on the all child classes and set the `parent` attribute on one of them.

```
<type name="Vendor\FreeDownload\Model\DownloadPdf"
  parent="Vendor\FreeDownload\Model\Download">
    <arguments>
        <argument name="tracking_level" xsi:type="number">4</argument>
...
<type name="Vendor\FreeDownload\Model\DownloadImage">
    <arguments>
        <argument name="tracking_level" xsi:type="number">4</argument>
...
```

C)

Configure the parameter on parent class, as it will be propagated on descendant classes.

```
<type name="Vendor\FreeDownload\Model\Download">
    <arguments>
        <argument name="tracking_level" xsi:type="number">4</argument>
    </arguments>
</type>
```

Options:

A- Option A

B- Option B

C- Option C

Answer:

B

Explanation:

To configure the tracking_level parameter in di.xml to have a value of 4 for the Download class and all classes that extend Download, the developer would use the following code:

<config>

```
<global>

<models>

<Vendor\FreeDownload\Model\Download>

<setting name='tracking_level' value='4'/>

</Vendor\FreeDownload\Model\Download>

<Vendor\FreeDownload\Model\DownloadPdf>

<rewrite name='tracking_level' value='4'/>

</Vendor\FreeDownload\Model\DownloadPdf>

<Vendor\FreeDownload\Model\DownloadImage>

<rewrite name='tracking_level' value='4'/>

</Vendor\FreeDownload\Model\DownloadImage>

</models>

</global>

</config>
```

The setting element is used to set a configuration value for a specific model. The rewrite element is used to override the default configuration value for a specific model. In this case, the tracking_level parameter is set to 4 for all models that extend Download.

Question 3

Question Type: MultipleChoice

An Adobe Commerce Developer wishes to add an action to a pre-existing route, but does not wish to interfere with the functionality of the actions from the original route.

What must the developer do to ensure that their action works without any side effects in the original module?

Options:

A- In the route declaration, use the before or after parameters to load their module in before or after the original module.

B- Inject the new action into the standard router constructor's $actiomist parameter.

C- Add the action into to the controllers/front_name/ in My.Module, Magento will automatically detect and use it.

Answer:

A

Explanation:

To add an action to a pre-existing route without interfering with the functionality of the original route, the developer must use the before or after parameters in the route declaration. This will load the developer's module in before or after the original module, respectively.

For example, the following code would add an action to the my_module/index route before the action from the original module:

<route id='my_module/index'>

<before>my_module_before</before>

</route>

The my_module_before action would be executed before the MyModule\Controller\Index action, which would allow the developer to perform any necessary setup before the original action is executed.

Question 4

Question Type: MultipleChoice

An Adobe Commerce developer is working on a Magento 2 instance which contains a B2C and a B2B website, each of which contains 3 different store views for English, Welsh, and French language users. The developer is tasked with adding a link between the B2C and B2B websites using a generic link template which is used throughout the sites, but wants these links to display in English regardless of the store view.

The developer creates a custom block for use with this template, before rendering sets the translate locale and begins environment emulation using the following code:

```
/** @var $this->_translate \Magento\Framework\TranslateInterface */
$this->_translate->setLocale($newLocaleCode);

/** @var $this->_emulation \Magento\Store\Model\App\Emulation */
$this->_emulation->startEnvironmentEmulation($storeId, \Magento\Framework\App\Area::AREA_FRONTEND);
```

They find that the template text is still being translated into each stores language. Why does this occur?

Options:

A- startEnvironmffntEmulation() SetS and locks the locale by Using the setLocale() Optional Second $lock parameter, i.e. setLocale($newLocaleCode,
true), to override and lock the locale of the emulated store. If this is set and locked initially then the environment emulation will not be able to override this.

B- startEnvironmentEmuiation() resets the translation locale to the one of the emulated stores, which overrides the locale the developer has set when the order of setLocate and startEnvironmentEmulation is used as displayed above.

C- setLocate() does not change translation locale after it has been initially set, the $this->_translate->emulate($newLocaiecode) method

exists to temporarily modify this by pushing the new locale to the top of the current emuiatedLocales stack.

Answer:

B

Explanation:

The startEnvironmentEmulation() method resets the translation locale to the one of the emulated stores, which overrides the locale the developer has set when the order of setLocale() and startEnvironmentEmulation() is used as displayed above.

The correct way to achieve the desired result is to use the emulate() method to temporarily modify the translation locale. The following code shows how to do this:

PHP

$this->_translate->emulate('en_US');

// Render the template

$this->_translate->revert();

This code will set the translation locale to English before rendering the template, and then revert the locale back to the default value after the template has been rendered.

The startEnvironmentEmulation() method is used to emulate a different store view or website. This can be useful for testing purposes, or for developing features that need to work in different environments.

The emulate() method is used to temporarily modify the translation locale. This can be useful for rendering templates in a specific language, or for testing features that need to work in different languages.

Question 5

Question Type: **MultipleChoice**

An Adobe Commerce developer has created a process that exports a given order to some external accounting system. Launching this process using the Magento CLI with the command php bin/magento my_module:order: process --order_id= is required.

Example: php bin/magento my_module:order:process --order_id=1245.

What is the correct way to configure the command?

A)

```
protected function configure()
{
    $this->setName('my_module:order:process');
    $this->setDescription('Processes an order');
    parent::configure();
}

protected function values()
{
    return [new InputValue('order_id', InputValue::REQUIRED, 'Order ID')];
}
```

B)

```
protected function configure()
{
    $this->setName('my_module:order:process');
    $this->setDescription('Processes an order');
    $this->addOption('order_id', null, InputOption::VALUE_REQUIRED, 'Order ID');
    parent::configure();
}
```

C)

```
protected function configure()
{
    $this->setName('my_module:order:process');
    $this->setDescription('Processes an order');
    $this->addOption('order_id', null, InputOption::VALUE_REQUIRED, 'Order ID');
    $this->setName('my_module:order:process');
    $this->setDescription('Processes an order');
    $this->addArgument('order_id', InputArgument::REQUIRED, 'Order ID');
    parent::configure();
}
```

D)

```
protected function configure()
{
    $this->setName('my_module:order:process');
    $this->setDescription('Processes an order');
    $this->addArgument('order_id', InputArgument::REQUIRED, 'Order ID');
    parent::configure();
}
```

Options:

A- Option B

B- Option C

C- Option C

D- Option D

Answer:

C

Explanation:

According to the How to use the Magento command-line interface (CLI) guide, to configure and run the Magento CLI, the developer needs to make the bin/magento file executable and then use it to run commands. To create a custom command, the developer needs to

create a class that implements \Symfony\Component\Console\Command\Command and define its name, description, arguments, options, and logic. The developer also needs to register the command in the di.xml file of their module using the Magento\Framework\Console\CommandList argument. Therefore, option C is the correct answer, as it shows the correct class and di.xml code to configure the custom command. Verified Reference: https://www.a2hosting.com/kb/installable-applications/optimization-and-configuration/magento1/using-the-magento-command-line-interface-cli/

Question 6

Question Type: MultipleChoice

An Adobe Commerce developer is tasked with adding custom data to orders fetched from the API. While keeping best practices in mind, how would the developer achieve this?

Options:

A- Create an extension attribute on Nagento\sales\Api\E)ata\orderinterface and an after plugin on Magento\Sales\Model\Order: :getExtensionAttributes() to add the custom data.

B- Create an extension attribute On Magento\Sales\Api\Data\OrderInterface and an after plugin On Magento\Sales\Api\OrderRepositoryInterface On geto and getListo to add the custom data.

C- Create a before plugin on Magento\sales\model\ResourceModel\order\collection: :load and alter the query to fetch the additional data.

Data will then be automatically added to the items fetched from the API.

Answer:

B

Explanation:

The developer should create an extension attribute on the Magento\Sales\Api\Data\OrderInterface interface and an after plugin on the Magento\Sales\Api\OrderRepositoryInterface::get() and Magento\Sales\Api\OrderRepositoryInterface::getList() methods.

The extension attribute will store the custom data. The after plugin will be used to add the custom data to the order object when it is fetched from the API.

Here is the code for the extension attribute and after plugin:

PHP

```
namespace MyVendor\MyModule\Api\Data;

interface OrderExtensionInterface extends \Magento\Sales\Api\Data\OrderInterface

{

/**

* Get custom data.
```

```
*
* @return string|null
*/
public function getCustomData();
/**
* Set custom data.
*
* @param string $customData
* @return $this
*/
public function setCustomData($customData);
}
namespace MyVendor\MyModule\Model;
class OrderRepository extends \Magento\Sales\Api\OrderRepositoryInterface
{
```

```
/**
* After get order.
*
* @param \Magento\Sales\Api\OrderRepositoryInterface $subject
* @param \Magento\Sales\Api\Data\OrderInterface $order
* @return \Magento\Sales\Api\Data\OrderInterface
*/
public function afterGetOrder($subject, $order)
{
if ($order instanceof OrderExtensionInterface) {
$order->setCustomData('This is custom data');
}
return $order;
}
/**
```

```
* After get list.
*
* @param \Magento\Sales\Api\OrderRepositoryInterface $subject
* @param \Magento\Sales\Api\Data\OrderInterface[] $orders
* @return \Magento\Sales\Api\Data\OrderInterface[]
*/
public function afterGetList($subject, $orders)
{
foreach ($orders as $order) {
if ($order instanceof OrderExtensionInterface) {
$order->setCustomData('This is custom data');
}
}
return $orders;
}
```

}

Once the extension attribute and after plugin are created, the custom data will be added to orders fetched from the API.

Question 7

Question Type: **MultipleChoice**

A logistics company with an Adobe Commerce extension sends a list of reviewed shipment fees to all its clients every month in a CSV file. The merchant then uploads this CSV file to a "file upload" field in admin configuration of Adobe Commerce.

What are the two requirements to display the "file upload" field and process the actual CSV import? (Choose two.)

A)

Add a custom backend model which extends `\Magento\Framework\App\Config\Value` and call `afterSave`:

```
// etc/adminhtml/system.xml
<field id="import_fees" ...>
    <label>Import shipment fees</label>
    <backend_model>My\Module\Model\Config\Backend\ImportFees</backend_model>

    ...

</field>
```

B)

```
// \My\Module\Model\Config\Backend\ImportFees

class \My\Module\Model\Config\Backend\ImportFees extends \Magento\Framework\App\Config\Value

...
public function afterSave()
{
    /** @var \My\Module\Model\ImportFeed $importFees */
    $importFees = $this->importFeesFactory->create();
    $importFees->uploadAndImport($this);
    return parent::afterSave();
}
```

C)

Add a new field in `etc/adminhtml/system.xml` in My_Module with the `file` type:

```
<field id="import_fees" translate="label" type="file" sortOrder="1000" showInDefault="1">
    <label>Import shipment fees</label>
    ...
</field>
```

D)

Add a new field in `etc/adminhtml/system.xml` in My_Module with a new custom type:

```
<field id="import_fees" translate="label" type="My\Module\Block\Adminhtml\Form\Field\ImportFees" sortOrder="1000"
    <label>Import shipment fees</label>
    ...
</field>
```

Options:

A- Option A

B- Option B

C- Option C

D- Option D

Answer:

A, B

Explanation:

To display the 'file upload' field and process the actual CSV import, the following two requirements must be met:

The developer must create a new system configuration setting that specifies the path to the CSV file.

The developer must create a new controller action that handles the file upload and import process.

The system.xml file is used to define system configuration settings. The following XML snippet shows how to define a new system configuration setting for the CSV file path:

XML

<config>

<system>

<config>

<shipment_fees_csv_path>/path/to/csv/file</shipment_fees_csv_path>

</config>

</system>

</config>

The Controller\Adminhtml\ShipmentFees controller class is used to handle the file upload and import process. The following code shows how to create a new controller action that handles the file upload and import process:

PHP

public function uploadAction()

{

$file = $this->getRequest()->getFile('shipment_fees_csv_file');

if ($file->isUploaded()) {

$importer = new ShipmentFeesImporter();

```
$importer->import($file);

}

return $this->redirect('adminhtml/system_config/edit/section/shipment_fees');

}
```

Question 8

Question Type: MultipleChoice

An Adobe Commerce developer was asked to provide additional information on a quote. When getting several quotes, the extension attributes are returned, however, when getting a single quote it fails to be returned.

What is one reason the extension attributes are missing?

Options:

A- The developer neglected to add coiiection='trueM to their attribute in etc/extension_attributes.xmi file. O ottribute code='my_attributesM type='MyVendor\MyModule\Api\Data\^AttributeInterface[]M collection='true' />

B- The developer neglected to provide a plugin On Hagento\Quote\Api\CartRepositoryInterface: :get.

C- The developer neglected to implement an observer on the coiiection_ioad_after event.

Answer:

A

Explanation:

The extension attributes are missing because the collection='true' attribute is not set in the etc/extension_attributes.xmi file. This attribute tells Magento that the extension attributes should be returned when the quote is retrieved.

To fix this issue, the developer needs to add the collection='true' attribute to the my_attributes extension attribute.

Once the collection='true' attribute is set, the extension attributes will be returned when the quote is retrieved.