Magento U

Exercise Solutions

Fundamentals of Magento 2 Development

Magento® U

training@magento.com

magento.com/training/overview

magento.com/training/catalog/certification

Contents

Unit One. Preparation & Configuration

Module 3. Magento 2 Overview

1.3.1. Create a new module. Make a mistake in its config. Create a second module dependent on the first.

Solution

1. Create a folder `app/code/Training/Test`.
   Create a file `app/code/Training/Test/etc/module.xml`:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="Training_Test" setup_version="2.0.0">
    </module>
</config>
```

2. Register your module with `app/code/Training/Test/ registration.php`

```
<?php
/**
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */

\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'Training_Test',
    __DIR__
);
```

3. Enable your module. Run `php bin/magento --clear-static-content module:enable Training_Test`
4. Run "bin/magento setup:upgrade" from the Magento root directory to upgrade your database.
   Make a mistake in the module.xml. For example, change `</module>` to `</mod>`. Then clean the cache (using the command `rf –rf var/*cache/*`) and load any page. You should get an error:
   Warning: DOMDocument::loadXML(): Opening and ending tag mismatch: module line 9 and mod in Entity, line: 10 in /var/www/magento/m2/lib/internal/Magento/Framework/Xml/Parser.php on line 159.
5. Fix the XML and clean the cache again: `php bin/magento cache:clean`
6. Create a folder `app/code/Training/Test2` and file `app/code/Training/Test2/etc/module.xml`:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
```

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="Training_Test" setup_version="2.0.0">
        <sequence>
            <module name="Magento_Test"/>
        </sequence>
    </module>
</config>
```

7. Enable your module. Run `php bin/magento --clear-static-content module:enable Training_Test2`
8. Clean the cache, and test whether your module is working.
9. You can disable `Training_Test` by setting its value to `0` in the `etc/config.php` or run `php bin/magento --clear-static-content module:disable Training_Test1`
10. After cleaning the cache, there will be no visible change. To see a list of loaded modules, go into the class `Magento\Framework\Module\ModuleList`, method `getNames()`, and put `print_r($result); exit;` before the return from the method. It will show a list of loaded methods, and you will see `Training_Test2` but no `Training_Test`.

Module 5. Development Operations

1.5.1: Mode

- Make sure the mode is set to Developer.
- Then, go into the `lib\internal\Magento\Framework\App\Bootstrap.php` file and throw an exception in the method, `run()`.
- See whether the exception is displayed on your screen. If it is, you have successfully set the mode. If not, review your steps.

(*It is possible that cache settings may need to be adjusted, but that topic is taught later in the course*).

1.5.2: Cache

Under what circumstances will cleaning the `var/cache` folder ***not*** work?

Answer: When cache storage is different (separate) from the `var/cache` folder. For example, it could be memory caching like Redis or Memcached.

Module 6. Dependency Injection & Object Manager

1.6.1: Dependency Injection

1. Go into the Magento core modules folder (`app/code/Magento`).
2. Open Catalog module; select 5 different classes from different folders.
3. What kind of a pattern do you notice?

Pattern: The constructor has a list of objects assigned to protected properties, and then used inside a class. This is what DI looks like in Magento 2

1.6.2: Object Manager

Go back to the two modules created you created in Exercise 1.3.1, `Unit1_FirstModule` and `Unit1_SecondModule*`.

In `Unit1_FirstModule`, create the folder “MagentoU”. In this folder, create the class “Test”. The code to be used is given below.

Note that the word “module” is not usually used in a module name. This is only used here for learning purposes.

Code for the class Test:

```
namespace Unit1\FirstModule\MagentoU;

class Test
{
      public function __construct(
            \Magento\Catalog\Api\ProductRepositoryInterface $productRepository,
            \Magento\Catalog\Model\ProductFactory $productFactory,
            \Magento\Checkout\Model\Session $session,
        \Unit1\FirstModule\Api\ProductRepositoryInterface $unit1ProductRepository,
            $justAParameter = false,
            array $data
      ) {

      }
}
```

Create the interface `Unit1\FirstModule\Api\ProductRepositoryInterface…` copy-paste its content from `Magento\Catalog\Api\ProductInterface.`

Now create the file `etc/di.xml.`

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <preference for="Unit1\FirstModule\Api\ProductRepositoryInterface"
type="Magento\Catalog\Model\ProductRepository" />
</config>
```

This will assign the `Magento\Catalog\Model\ProductRepository` class to `Unit1\FirstModule\Api\ProductRepositoryInterface`.

Next, modify the parameters of a constructor:

```
    <type name="Unit1\FirstModule\MagentoU\Test">
        <arguments>
            <argument name="justAParameter" xsi:type="string”> Hello world! </argument>
<argument name=”data” type=”array”>
    <item name=”test-array-item”>Test Array Item!!!</item>
</argument>
        </arguments>
    </type>
```

You have now added a new element to the object’s constructor.

Finally, create a `di.xml` file in the `SecondModule` and add another item to the `$data` array from there.

Module 7. Plugins

1.7.1: Plugins 1

Although you do not commonly interact with Interceptors, it is useful to understand how plugins work. This can be helpful in the debugging process.

In your Magento installation, take a look at a couple of interceptors and note how similar they are to each other – for example, `var/generation/Magento/Catalog/Model/Product/Interceptor.php`.

1.7.2: Plugins 2

For the class... `Magento\Catalog\Model\Product` and the method... `getPrice()`:

Create a plugin that will modify price (`afterPlugin`).

1. Customize `Magento\Theme\Block\Html\Footer` class, to replace the body of the `getCopyright()` method with your implementation. Return a hard-coded string: “Customized copyright!”

2. Customize `Magento\Theme\Block\Html\Breadcrumbs` class, `addCrumb()` method, so that every `crumbName` is transformed into:
`$crumbName . “(!)”`

Module 8. Events

1.8.1. In your module, create an observer to the event `controller_action_predispatch`. Get the URL from the request object `request->getPathInfo()`. Log it into the file.

Note that if you decide to extend the XML config file, you will also need to update the XSD schema as well.

Solution

1. Create an event declaration in `events.xml`:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Event/etc/events.xsd">
    <event name="controller_action_predispatch">
        <observer name="training_test"
instance="Training\Test\Observer\RedirrectToProductView" shared="false" />
    </event>
</config>
```

2. Create an Observer:

```
<?php
namespace Training\Test\Observer;

use Magento\Framework\Event\ObserverInterface;

class RedirrectToProductView implements ObserverInterface
{
    /**
     * @var \Magento\Framework\App\Response\RedirectInterface
     */
    protected $redirect;

    /**
     * @var \Magento\Framework\App\ActionFlag
     */
    protected $_actionFlag;

    /**
     * @param \Magento\Framework\App\Response\RedirectInterface $redirect
     */
    public function __construct(
        \Magento\Framework\App\Response\RedirectInterface $redirect,
        \Magento\Framework\App\ActionFlag $actionFlag
    ) {
        $this->redirect = $redirect;
        $this->_actionFlag = $actionFlag;
    }

    /**
     *
     * @param \Magento\Framework\Event\Observer $observer
     * @return $this
     *
     * @SuppressWarnings(PHPMD.UnusedLocalVariable)
     */
    public function execute(\Magento\Framework\Event\Observer $observer) {
        $request = $observer->getEvent()->getData('request');
        if ($request->getModuleName() != 'catalog' || $request->getControllerName() !=
'product') {
            $controller = $observer->getControllerAction();
            $this->_actionFlag->set('',
\Magento\Framework\App\Action\Action::FLAG_NO_DISPATCH, true);
            $this->redirect->redirect($controller->getResponse(),
'catalog/product/view/id/1');
        }
    }
}
```

Result: Now all pages redirect to catalog/product/view.

Module 9. Module Configuration

1.9.1. In the empty module you created in Exercise 1.3.1, add custom configuration xml/xsd files.

To create new xml/xsd files, we have to take the following steps:

Phase 1: Create `test.xml` and `test.xsd` files.
Phase 2: Create PHP files to process them: `Config`, `ConfigInterface`, `Convertor`, `Reader`, `SchemaLocator`.
Phase 3: Define a preference for `ConfigInterface`.
Phase 4: Test: In this example we will create a new controller to test this functionality.

Let's follow through each step.

Phase 1

1.1) Create `etc/test.xml`:

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="test.xsd">
    <mynode>HELLO</mynode>
    <mynode>HELLO 2</mynode>
</config>
```

1.2) Create `etc/test.xsd`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"> <xs:element name="config">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="mynode" type="xs:string" maxOccurs="10"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

Phase 2

2.1) Create an interface:

```
<?php
namespace Training\Test\Model\Config;

interface ConfigInterface {
    public function getMyNodeInfo();
}
```

2.2) Create Config class:

```
<?php
namespace Training\Test\Model;
```

```
class Config extends \Magento\Framework\Config\Data implements
\Training\Test\Model\Config\ConfigInterface
{
    public function __construct(
       \Training\Test\Model\Config\Reader $reader,
       \Magento\Framework\Config\CacheInterface $cache,
       $cacheId = 'training_test_config'
    ) {
            parent::__construct($reader, $cache, $cacheId);
    }

    public function getMyNodeInfo() {
       return $this->get();
    }
}
```

2.3) Create Reader class:

```
<?php
namespace Training\Test\Model\Config;

class Reader extends \Magento\Framework\Config\Reader\Filesystem
{
    /**
     * List of id attributes for merge
     *
     * @var array
     */
    protected $_idAttributes = []; //['/config/option' => 'name', '/config/option/inputType'
=> 'name'];

    /**
     * @param \Magento\Framework\Config\FileResolverInterface $fileResolver
     * @param \Magento\Catalog\Model\ProductOptions\Config\Converter $converter
     * @param \Magento\Catalog\Model\ProductOptions\Config\SchemaLocator
     * $schemaLocator
     * @param \Magento\Framework\Config\ValidationStateInterface $validationState
     * @param string $fileName
     * @param array $idAttributes
     * @param string $domDocumentClass
     * @param string $defaultScope
     */

public function __construct(
       \Magento\Framework\Config\FileResolverInterface $fileResolver,
       \Training\Test\Model\Config\Converter $converter,
       \Training\Test\Model\Config\SchemaLocator $schemaLocator,
       \Magento\Framework\Config\ValidationStateInterface $validationState,
       $fileName = 'test.xml',
       $idAttributes = [],
```

```
        $domDocumentClass = 'Magento\Framework\Config\Dom',
        $defaultScope = 'global'
    ) {
             parent::__construct(
            $fileResolver,
            $converter,
            $schemaLocator,
            $validationState,
            $fileName,
            $idAttributes,
            $domDocumentClass,
            $defaultScope
        );
    }
}
```

2.4) Create schemaLocator class:

```
<?php
namespace Training\Test\Model\Config;

class SchemaLocator implements \Magento\Framework\Config\SchemaLocatorInterface
{
    /**
     * Path to corresponding XSD file with validation rules for merged config
     *
     * @var string
     */
    protected $_schema = null;

    /**
     * Path to corresponding XSD file with validation rules for separate config
     * files
     * @var string
     */
    protected $_perFileSchema = null;

    /**
     * @param \Magento\Framework\Module\Dir\Reader $moduleReader
     */
    public function __construct(\Magento\Framework\Module\Dir\Reader $moduleReader)
    {
        $etcDir = $moduleReader->getModuleDir('etc', 'Training_Test');
        $this->_schema        = $etcDir . '/test.xsd';
        $this->_perFileSchema = $etcDir . '/test.xsd';
    }

    /**
     * Get path to merged config schema
     *
```

```
     * @return string|null
     */
    public function getSchema()
    {
        return $this->_schema;
    }

    /**
     * Get path to pre file validation schema
     *
     * @return string|null
     */
    public function getPerFileSchema()
    {
       return $this->_perFileSchema;
    }
}
```

2.5) Create converter class:

```
<?php

namespace Training\Test\Model\Config;

class Converter implements \Magento\Framework\Config\ConverterInterface
{
    /**
     * Convert dom node tree to array
     *
     * @param \DOMDocument $source
     * @return array
     * @throws \InvalidArgumentException
     */
    public function convert($source)
    {
        $output = [];

        /** @var $optionNode \DOMNode */
       foreach ($source->getElementsByTagName('mynode') as $node) {
            $output[] = $node->textContent;
        }
        return $output;
    }
}
```

Phase 3

Set a preference in the `di.xml`:

```
 <preference for="Training\Test\Model\Config\ConfigInterface"
type="Training\Test\Model\Config" />
```

Phase 4

4.1) Create a controller file (assuming you've set up routes.xml already):

```
<?php
/**
 * Product controller.
 *
 * @copyright Copyright (c) 2014 X.commerce, Inc. (http://www.magentocommerce.com)
 */
namespace Training\Test\Controller\Action;

class Config extends \Magento\Framework\App\Action\Action
{

    public function execute() {
        $testConfig = $this->_objectManager-
>get('Training\Test\Model\Config\ConfigInterface');
        $myNodeInfo = $testConfig->getMyNodeInfo();
        if (is_array($myNodeInfo)) {
            foreach($myNodeInfo as $str) {
                $this->getResponse()->appendBody($str . "<BR>");
            }
        }
    }
}
```

4.2) Hit a page /test/action/config. You will see:

HELLO
HELLO 2

Unit Two. Request Flow

Module 2. Request Flow Overview

2.2.1. Find a place in the code where output is flushed to the browser. Create an extension that captures and logs the file-generated page HTML. (“Flushing output” means a “send” call to the response object.)

Solution

1. Declare an event in the file `etc/frontend/events.xml`:

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Event/etc/events.xsd">
    <event name="controller_front_send_response_before">
        <observer name="training_test" instance="Training\Test\Observer\LogPageOutput"
shared="false" />
    </event>
</config>
```

2. Create an observer class:

```
<?php
namespace Training\Test\Observer;

use Magento\Framework\Event\ObserverInterface;

class LogPageOutput implements ObserverInterface
{
    /**
     * @var null|\Psr\Log\LoggerInterface
     */
    protected $_logger = null;

    /**
     * @param \Psr\Log\LoggerInterface $logger
     */
    public function __construct(\Psr\Log\LoggerInterface $logger)
    {
        $this->_logger = $logger;
    }
    /**
     *
     * @param \Magento\Framework\Event\Observer $observer
     * @return $this
     *
     * @SuppressWarnings(PHPMD.UnusedLocalVariable)
     */
    public function execute(\Magento\Framework\Event\Observer $observer) {
        $response = $observer->getEvent()->getData('response');
        $body = $response->getBody();
        $this->_logger->addDebug("--------\n\n\n BODY \n\n\n ". $body);
    }
```

Module 3. Request Routing

2.3.1. Create an extension that logs into the file list of all available routers into a file.

Solution

1. Create a preference in the `di.xml`:

```
<preference for="Magento\Framework\App\FrontControllerInterface"
type="Training\Test\App\FrontController" />
```

2. Implement a front controller class:

```
<?php
namespace Training\Test\App;

class FrontController extends \Magento\Framework\App\FrontController
{
    /**
     * @var \Magento\Framework\App\RouterList
     */
    protected $routerList;

    /**
     * @var \Magento\Framework\App\Response\Http
     */
    protected $response;

    /**
     * @var \Psr\Log\LoggerInterface
     */
    protected $logger;

    /**
     * @param \Magento\Framework\App\RouterList $routerList
     * @param \Magento\Framework\App\Response\Http $response
     * @param \Psr\Log\LoggerInterface $logger
     */
    public function __construct(
        \Magento\Framework\App\RouterList $routerList,
        \Magento\Framework\App\Response\Http $response,
        \Psr\Log\LoggerInterface $logger)
    {
        $this->routerList = $routerList;
        $this->response = $response;
        $this->logger = $logger;
    }

    /**
     * @param \Magento\Framework\App\RequestInterface $request
     * @return
\Magento\Framework\App\ResponseInterface|\Magento\Framework\Controller\ResultInterface
     */
    public function dispatch(\Magento\Framework\App\RequestInterface $request) {
        foreach ($this->routerList as $router) {
```

```
            $this->logger->addDebug(get_class($router));
        }
        return parent::dispatch($request);
    }
}
```

2.3.2. Create a new router which "understands" URLs like /frontName-actionPath-action and converts them to /frontName/actionPath/action

Solution

1. Declare your router. Add the following code to the etc/frontend/di.xml of your module (assuming your module is Training_Test):

```
<type name="Magento\Framework\App\RouterList">
    <arguments>
        <argument name="routerList" xsi:type="array">
            <item name="training" xsi:type="array">
                <item name="class"
                     xsi:type="string">Training\Test\Controller\Router</item>
                <item name="disable" xsi:type="boolean">false</item>
                <item name="sortOrder" xsi:type="string">70</item>
            </item>
        </argument>
    </arguments>
</type>
```

2. Create a router class:

```
<?php

namespace Training\Test\Controller;

class Router implements \Magento\Framework\App\RouterInterface
{
    public function __construct(\Magento\Framework\App\ActionFactory $actionFactory) {
        $this->actionFactory = $actionFactory;
    }

    public function match(\Magento\Framework\App\RequestInterface $request) {
        $info = $request->getPathInfo();

        if (preg_match("%^/(test)-(.*?)-(.*?)$%", $info, $m)) {
            $request->setPathInfo(sprintf("/%s/%s/%s", $m[1], $m[2], $m[3]));
            return $this->actionFactory->create('Magento\Framework\App\Action\Forward',
                 ['request' => $request]);
                 }
        return null;
    }
}
```

In this example, the router only “understands” urls that start with “test”. To make it work with every url, remove the line:

```
if (preg_match("%^/(test)-(.*?)-(.*?)$%", $info, $m)) {
```

2.3.3. Modify Magento so a “Not Found” page will forward to the home page.

Solution

There are many different ways to do this. The easiest is to change the config option /web/default/noroute. This will change the 404 page for all requests. To make the code more flexible, you can create a new NoRouteHandler. To do this:

1. Declare your handler in the di.xml:

```
<type name="Magento\Framework\App\Router\NoRouteHandlerList">
    <arguments>
        <argument name="handlerClassesList" xsi:type="array">
            <item name="default" xsi:type="array">
                <item name="class"
                        xsi:type="string">Training\Test\Controller\NoRouteHandler</item>
                <item name="sortOrder" xsi:type="string">200</item>
            </item>
        </argument>
    </arguments>
</type>
```

2. Create a handler class:

```
<?php
namespace Training\Test\Controller;

class NoRouteHandler implements \Magento\Framework\App\Router\NoRouteHandlerInterface {

    public function process(\Magento\Framework\App\RequestInterface $request) {
        $moduleName      = 'cms';
        $controllerName  = 'index';
        $actionName      = 'index';

        $request
          ->setModuleName($moduleName)
          ->setControllerName($controllerName)
          ->setActionName($actionName);
        return true;
    }
}
```

Module 5. Working with Controllers

2.5.1. Create a frontend controller that renders “HELLO WORLD”

Solution

1. Declare a route in `etc/frontend/routes.xml`:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:App/etc/routes.xsd">
    <router id="standard">
        <route id="test" frontName="test">
            <module name="Training_Test" />
        </route>
    </router>
</config>
```

2. Create an action class:

```
<?php
/**
 * Product controller.
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Training\Test\Controller\Action;

class Index extends \Magento\Framework\App\Action\Action
{
    public function execute() {
        $this->getResponse()->appendBody("HELLO WORLD");
    }
}
```

2.5.2. Customize the catalog product view controller using plugins and preferences.

Solution

1. To add a plugin or preference, use the following code in `di.xml`:

```
<preference for="Magento\Catalog\Controller\Product\View"
 type="Training\Test\Controller\Product\View" />
```

or

```
<type name="Magento\Catalog\Controller\Product\View">
 <plugin name="product-view-controller-plugin"
         type="Training\Test\Controller\Product\View" sortOrder="10"/>
</type>
```

Note: You will create a preference ***or*** plugin within one module.

2. Now you can implement your preference/plugin:

```
<?php

namespace Training\Test\Controller\Product;

class View extends \Magento\Framework\App\Action\Action
{
    /**
    public function execute() {
        echo "ONE"; exit;
    }
    public function beforeExecute() {
        //echo "BEFORE<BR>"; exit;
    }
    public function afterExecute(\Magento\Catalog\Controller\Product\View $controller,
$result) {
        //echo "AFTER"; exit;
    }
    */
}
```

3. Uncomment the appropriate method for testing: Uncomment "`execute`" for preferences, and "`beforeExecute`", "`afterExecute`" for plugins.

2.5.3. Create an adminhtml controller that allows access only if the GET parameter "secret" is set.

Solution

1. Create a file `etc/adminhtml/routes.xml`:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:App/etc/routes.xsd">
    <router id="admin">
        <route id="adminhtml">
            <route id="test" frontName="test">
                <module name="Training_Test" before="Magento_Adminhtml" />
            </route>
        </route>
    </router>
</config>
```

2. Create an action class:

```php
<?php
/**
 * Product controller.
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Training\Test\Controller\Action;

class Index extends \Magento\Framework\App\Action\Action
{
    /**
     * Test action index
     */
    public function execute() {
        $this->getResponse()->appendBody("HELLO WORLD");
    }

   /**
    * Check if admin has permissions to visit related pages
    *
    * @return bool
    */
    protected function _isAllowed() {
        $secret = $this->getRequest()->getParam('secret');
        return isset($secret) && (int)$secret==1;
    }
}
```

2.5.4. Make the “Hello World” controller you just created redirect to a specific category page.

Solution

Put a line `$this->_redirect('catalog/category/view/id/_CATEGORY_ID_')` into the execute method (but replace `_CATEGORY_ID_` with the real category_id).

Module 6. URL Rewrites

2.6.1. Create a URL rewrite for the “Hello World” controller.

Solution

Add one record to the url_rewrite table:

```
INSERT INTO url_rewrite SET request_path='testpage.html', target_path='test/action/index',
redirect_type=0, store_id=1, is_autogenerated=0;
```

Unit Three. Rendering

Module 3. Rendering Flow

3.3.1. In the core files, find and print out the layout XML for the product view page.

Solution

```
\Magento\Framework\View\Layout::generateXml()
```

Module 5. Block Architecture & Life Cycle

3.5.1. Create a block extending `AbstractBlock` and implement the `_toHtml()` method. Render that block in the new controller.

Solution

1. Create the block:

```
<?php
namespace Training\Test\Block;

class Test extends \Magento\Framework\View\Element\AbstractBlock
{
    protected function _toHtml() {
        return "<b>Hello world from block!</b>";
    }
}
```

2. Create an action class:

```
<?php
namespace Training\Test\Controller\Block;

class Index extends \Magento\Framework\App\Action\Action
{

    public function execute() {
        $layout = $this->_view->getLayout();
        $block = $layout->createBlock('Training\Test\Block\Test');
      $this->getResponse()->appendBody($block->toHtml());
    }
}
```

3.5.2. Create and render a text block in the controller.

Solution

Create an action class:

```
<?php

namespace Training\Test\Controller\Block;

class Text extends \Magento\Framework\App\Action\Action
{
    public function execute() {
        $block = $this->_view->getLayout()-
>createBlock('Magento\Framework\View\Element\Text');
        $block->setText("Hello world from text block !");
       $this->getResponse()->appendBody($block->toHtml());
    }
}
```

3.5.3. Customize the `Catalog\Product\View\Description` block, implement the `_beforeToHtml()` method, and set a custom description for the product here.

Solution

1. Declare a plugin in the etc/frontend/di.xml:

```
    <type name="Magento\Catalog\Block\Product\View\Description">
        <plugin name="product-view-desription-plugin"
                type="Training\Test\Block\Product\View\Description" sortOrder="10"/>
    </type>
```

2. Create a plugin class:

```
<?php
namespace Training\Test\Block\Product\View;

class Description extends \Magento\Framework\View\Element\Template
{
    public function beforeToHtml(\Magento\Catalog\Block\Product\View\Description
$originalBlock) {
        $originalBlock->getProduct()->setDescription('Test description');
    }
}
```


Module 6. Templates

3.6.1. Define which template is used in `Catalog\Block\Product\View\Attributes`.

Solution

`Magento/Catalog/view/frontend/templates/product/view/attributes.phtml`

3.6.2. Create a template block and a custom template file for it. Render the block in the controller.

Solution

1. Create the block:

```php
<?php

namespace Training\Test\Block;

class Template extends \Magento\Framework\View\Element\Template
{
}
```

Note: You cannot create your own block. You must use Magento\Framework\View\Element\Template, since it is not an abstract.

2. Create a template file `Training/Test/view/frontend/test.phtml`:

   "`Hello from template`".

3. Create an action class:

```php
<?php
/**
 * Product controller.
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Training\Test\Controller\Action;

class Index extends \Magento\Framework\App\Action\Action
{
    public function execute() {
        $block = $this->_view->getLayout()->createBlock('Training\Test\Block\Test');
        $block->setTemplate('test.phtml');
        $this->getResponse()->appendBody($block->toHtml());

    }
}
```

3.6.3. Customize the `Catalog\Block\Product\View\Description` block and assign a custom template to it.

Solution

1. Using the same declaration as in 2.3, change the `beforeToHtml` method to:

```
public function beforeToHtml(\Magento\Catalog\Block\Product\View\Description $originalBlock)
{
        $originalBlock->setTemplate('Training_Test::description.phtml');
    }
```

2. Create a template `Training/Test/view/frontend/templates/description.phtml`:

```
<h1>Custom description template!</h1>
```

Module 8. Layout XML: Loading & Rendering

You will be provided with a code archive containing the solutions for the exercises in this module.

3.8.1. Add a `default.xml` layout file to the `Training_Render` module.

1. Reference the `content.top` container.
2. Add a `Magento\Framework\View\Element\Template` block with a custom template.
3. Create your custom template.
4. Check that the template content is visible on every page.

3.8.2. Create a new controller action (ex: `training_render/layout/onepage`).

1. For that action, choose a single-column page layout using layout XML.
2. Set a page title using layout XML.

3.8.3. Add an `arguments/argument` node to the block.

1. Set the argument name to `background_color`.
2. Set the argument value to `lightskyblue`.
3. In the template, add an inline style attribute to a `<div>` element:

```
style="background_color:
<?= $this->getData('background_color') ?>;"
```

4. Confirm that the background color is displayed.

3.8.4. Change the block color to orange on the product detail pages only.

3.8.5. On category pages, move the exercise block to the bottom of the left column.

3.8.6. On the custom action you just added, remove the custom block from the `content.top` container. (See Exercise 3.8.1.)

3.8.7. Using layout XML, add a new link for the custom page you just created to the set of existing links at the top of every page.

Unit Four. Databases & Entity-Attribute-Value (EAV)

Module 2. Databases Overview

List Root Categories by Store

4.2.1. Echo the list of all store views and associated root categories.

1. Get a list of stores using: `Magento\Store\Model\ResourceModel\Store\Collection`
2. Get root category IDs using:

   `Magento\Store\Model\Store::getRootCategoryId()`
3. Create a category collection and filter it by the root category IDs.
4. Add the category name attribute to the result.
5. Display stores with the associated root category names.

Solution

You will be provided with a code archive containing the solutions for the exercises in this module.

Module 3. Models Detailed Workflow

You will be provided with a code archive containing the solutions for the exercises in this module.

4.3.1. Product Save Operations

1. Log every product save operation.
2. Specify the `productId` and the data that has been changed.

Module 4. Setup Scripts & Setup Resources

You will be provided with a code archive containing the solutions for the exercises in this module.

4.4.1. Create a table with an install script for the module `Training_Vendor`.

1. Give `Training_Vendor` a `setup_version` of `0.0.1`.
2. Create the `Setup` folder.
3. Create the `InstallSchema` class.
4. Create a `training_vendor_entity` table using DDL methods.
5. Execute the installation using the console tool.
6. Verify that it works by checking the `setup_module` table.

4.4.2. Create a regular upgrade script to add an additional column.

1. Create the `UpgradeSchema` class.
2. Add an additional column to the `training_vendor_entity` table using DDL adapter methods.

3. Upgrade the version number in your `module.xml` to `0.0.2`.
4. Run the appropriate console command.
5. Verify that it works.

4.4.3. Create a data upgrade script to set a config value.

1. Create the `UpgradeData` class.
2. Define a fixture `vendor` to be installed along with your module.
3. Upgrade the module version.
4. Execute the appropriate console command.
5. Verify that it works.

Module 7. Attribute Management

4.7.1. Create a text input attribute (1) from the Admin interface.

1. Add it to an attribute set.
2. Check that it appears on the product edit page.
3. Make it visible on the storefront product view page.

Solution

Note that this exercise does not require any coding. It must be completed using the browser to access the Magento Admin backend and the storefront only.

1. Log in to the Magento backend.
2. Select **Stores > Attribute > Product** in the main navigation.
3. Click the **Add New Attribute** button.
4. Enter an attribute label, for example: **Flavor.**
5. Select the tab **Frontend Properties.**
6. Set **Visible on Catalog Pages on Frontend** to **Yes.**
7. Click **Save Attribute.**
8. Select **Stores > Attribute > Product Templates** in the main navigation.
9. In the list, select an Attribute Set, for example the **Bag** attribute set.
10. Drag & drop the **Flavor** attribute icon from the right **Unassigned Attributes** column onto the **Product Details** attribute group folder icon.
11. Confirm that the **Flavor** attribute icon now is listed within the **Product Details** attribute group.
12. Click **Save Attribute Set.**
13. Select **Products > Inventory > Catalog in** the main navigation.
14. In the **Attribute Set** column filter dropdown, select **Default** and click the **Search** button.
15. Select a product from the list where the **Visibility** is set to **Catalog, Search,** for example the **Push It Messenger Bag.**
16. Confirm that the **Flavor** attribute field is displayed on the **Product Details** form.
17. Enter a value for the **Flavor** attribute, for example **Strawberry.**
18. Click the **Save** button.
19. Open the product in the Magento storefront.
20. Select the **Additional Information** tab.
21. Confirm that the new attribute and the value you gave it are displayed.

4.7.2. Create a text input attribute (2) from an install data method.

Follow the steps in the previous exercise and create a text input attribute that is visible on the storefront, but this time create it from an install data method.

Solution

The exercise solution requires coding.

1. Create a module `Training_Orm`. Create an `InstallData` setup class.

```php
<?php
namespace Training\Orm\Setup;

use Magento\Catalog\Model\Product;
use Magento\Catalog\Model\ResourceModel\Eav\Attribute as CatalogAttribute;
use Magento\Catalog\Setup\CategorySetup;
use Magento\Catalog\Setup\CategorySetupFactory;
use Magento\Framework\Setup\InstallDataInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\ModuleDataSetupInterface;

class InstallData implements InstallDataInterface
{
    /**
     * @var CategorySetupFactory
     */
    private $catalogSetupFactory;

    public function __construct(CategorySetupFactory $categorySetupFactory)
    {
        $this->catalogSetupFactory = $categorySetupFactory;
    }

    /**
     * Installs data for a module
     *
     * @param ModuleDataSetupInterface $setup
     * @param ModuleContextInterface $context
     * @return void
     */
    public function install(ModuleDataSetupInterface $setup, ModuleContextInterface $context)
    {
    }
}
```

2. In the install method, create an instance of the `CategorySetup` class and call `addAttribute` on it.

```php
public function install(ModuleDataSetupInterface $setup, ModuleContextInterface $context)
{
    /** @var CategorySetup $catalogSetup */
    $catalogSetup = $this->catalogSetupFactory->create(['setup' => $setup]);
    $catalogSetup->addAttribute(Product::ENTITY, 'flavor_from_setup_method', [
        'label' => 'Flavor From Setup Method',
        'visible_on_front' => 1,
        'required' => 0,
```

```
            'global' => CatalogAttribute::SCOPE_STORE
        ]);
    }
```

3. Enable the module and apply the setup changes.

```
php bin/magento --clear-static-content
module:enable Training_Orm

php bin/magento setup:upgrade
```

4. Open a product in the Admin interface and confirm that the new attribute exists and can be set on a store view level.
5. Visit a product on the storefront and confirm that the new attribute is visible there, too.

4.7.3. Create a multiselect product attribute from an upgrade data method.

1. Create a multiselect product attribute.
2. Set the `backend_model` property to `Magento\Eav\Entity\Attribute\Backend\ArrayBackend`
3. Add a few options to the attribute.
4. Make it visible in the catalog product view page.

4.7.4. Customize the frontend rendering of the attribute values.

1. Customize the rendering of the values of the multiselect product attribute that you created in the previous exercise.
2. Show it as an HTML list rather than as comma-separated values.

Solution

The exercise solution requires coding.

1. Expand the `Training_Orm` module by adding a new section to the `UpgradeData` setup class.

```
public function upgrade(
    ModuleDataSetupInterface $setup,
    ModuleContextInterface $context
) {
    $dbVersion = $context->getVersion();

    if (version_compare($dbVersion, '0.1.1', '<')) {
        // ...
    }
    if (version_compare($dbVersion, '0.1.2', '<')) {
        /** @var CategorySetup $catalogSetup */
        $catalogSetup = $this->catalogSetupFactory->create(['setup' => $setup]);
        $catalogSetup->updateAttribute(
            Product::ENTITY,
```

```
            'example_multiselect',
            [
                'frontend_model' =>
                    \Training\Orm\Entity\Attribute\Frontend\HtmlList::class,
                'is_html_allowed_on_front' => 1,
            ]
        );
    }
}
```

2. Update the setup version in the `module.xml` file to `0.1.2`.

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../../lib/internal/Magento/Framework/Module/etc/module.xsd">
    <module name="Training_Orm" setup_version="0.1.2">
        <sequence>
            <module name="Magento_Eav"/>
        </sequence>
    </module>
</config>
```

3. Create the HtmlList frontend model.

```
<?php


namespace Training\Orm\Entity\Attribute\Frontend;

class HtmlList extends
    \Magento\Eav\Model\Entity\Attribute\Frontend\AbstractFrontend
{
    /**
     * @param \Magento\Framework\Object $object
     * @return string
     */
    public function getValue(\Magento\Framework\Object $object)
    {
        if ($this->getConfigField('input') !== 'multiselect') {
            return parent::getValue($object);
        }
        return $this->getValuesAsHtmlList($object);
    }

    /**
     * @param \Magento\Framework\Object $object
     * @return string
     */
```

```
    private function getValuesAsHtmlList(\Magento\Framework\Object $object)
    {
        $options = $this->getOptions($object);
        $escapedOptions = array_map('htmlspecialchars', $options);
        return sprintf(
            '<ul><li>%s</li></ul>',
            implode('</li><li>', $escapedOptions)
        );
    }

    /**
     * @param \Magento\Framework\Object $object
     * @return string[]
     */
    private function getOptions(\Magento\Framework\Object $object)
    {
        $optionId = $object->getData($this->getAttribute()->getAttributeCode());
        $option = $this->getOption($optionId);
        return $this->isSingleValue($option) ? [$option] : $option;
    }

    /**
     * @param string[]|string $option
     * @return bool
     */
    private function isSingleValue($option)
    {
        return !is_array($option);
    }
}
```

4. Run the setup upgrade process.
5. Set both options on a product and save it.
6. View the product on the storefront and confirm the option values are rendered as an HTML list.

4.7.5. Create a select attribute with a predefined list of options.

1. Create a new customer attribute 'priority' using an upgrade data method.
   - Use the frontend_input type 'select'.
   - Use the `backend_type` 'int'.
   - Set `is_system` to 0.
2. Assign a custom source model.
3. Implement the custom attribute source model to list numbers from 1 through 10.
4. Test that the attribute works as expected.

Solution

The exercise solution requires coding.

1. Expand the `Training_Orm` module by adding a new dependency on `Magento\Customer\Setup\CustomerSetupFactory` to the `DataUpgrade` class. Initialize the field using dependency injection.

```
// ...
use Magento\Customer\Setup\CustomerSetup;
use Magento\Customer\Setup\CustomerSetupFactory;

// ...

public function __construct(
    CategorySetupFactory $categorySetupFactory,
    CustomerSetupFactory $customerSetupFactory,
    StoreManager $storeManager
) {
    $this->catalogSetupFactory = $categorySetupFactory;
    $this->customerSetupFactory = $customerSetupFactory;
    $this->storeManager = $storeManager;
}
// ...
```

2. Add a new section to the `UpgradeData` setup class.

```
public function upgrade(
    ModuleDataSetupInterface $setup,
    ModuleContextInterface $context
) {
    $dbVersion = $context->getVersion();

    if (version_compare($dbVersion, '0.1.1', '<')) {
        // ...
    }
    if (version_compare($dbVersion, '0.1.2', '<')) {
        // ...
    }
    if (version_compare($dbVersion, '0.1.3', '<')) {
        /** @var CustomerSetup $customerSetup */
        $customerSetup = $this->customerSetupFactory->create(['setup' => $setup]);
        $customerSetup->addAttribute(
            Customer::ENTITY,
            'priority',
            [
                'label' => 'Priority',
                'type' => 'int',
                'input' => 'select',
                'source' => \Training\Orm\Entity\Attribute\Source\CustomerPriority::class,
                'required' => 0,
                'system' => 0,
                'position' => 100
            ]
```

```
        );
        $customerSetup->getEavConfig()->getAttribute('customer', 'priority')
            ->setData('used_in_forms', ['adminhtml_customer'])
            ->save();
    }
}
```

3. Note that the system property must be set to 0 for Magento to recognize the custom attribute when saving a customer through the Admin interface.
4. Note that customer attributes must be added to the customer_form_attribute explicitly to be visible in the browser.
5. Create the custom source model.

```
<?php

namespace Training\Orm\Entity\Attribute\Source;

use Magento\Eav\Model\Entity\Attribute\Source\AbstractSource;

class CustomerPriority extends AbstractSource
{
    /**
     * Retrieve All options
     *
     * @return array[]
     */
    public function getAllOptions()
    {
        $options = array_map(function($priority) {
            return [
                'label' => sprintf('Priority %d', $priority),
                'value' => $priority
            ];
        }, range(1, 10));
        if ($this->getAttribute()->getFrontendInput() === 'select') {
            array_unshift($options, ['label' => '', 'value' => 0]);
        }
        return $options;
    }
}
```

6. Run the setup scripts.
7. Visit a customer in the Admin interface, and select the **Account Information** tab to confirm that the field is visible and that the list of priorities is visible.
8. Set a value for the Priority attribute and save the customer to confirm that the value is being saved on the entity.

Unit Five. Service Contracts

Module 4. Services API: Repositories & Business Logic

Use the native customer and product repository classes to obtain lists of objects.

Preparation

1. Create a new module `Training_Registry` for the exercise solutions.

2. Add a frontend route configuration.

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../../../lib/internal/Magento/Framework/App/etc/routes.xsd">
    <router id="standard">
        <route id="training_repository" frontName="training_repository">
            <module name="Training_Repository"/>
        </route>
    </router>
</config>
```

5.4.1. Obtain a list of products via the product repository.

- Print a list of products.
- Add a filter to the search criteria.
- Add another filter with a logical AND condition.
- Add a sort order instruction.
- Limit the number of products to 6.

Solution

1. Create an action controller to output the exercise result.

```
<?php

namespace Training\Repository\Controller\Repository;

use Magento\Catalog\Api\Data\ProductInterface;
use Magento\Catalog\Api\ProductRepositoryInterface;
use Magento\Framework\Api\SearchCriteriaBuilder;
use Magento\Framework\App\Action\Action;
use Magento\Framework\App\Action\Context;

class Product extends Action
{
    /**
     * @var ProductRepositoryInterface
     */
```

```
    private $productRepository;

    /**
     * @var SearchCriteriaBuilder
     */
    private $searchCriteriaBuilder;

    public function __construct(
        Context $context,
        ProductRepositoryInterface $productRepository,
        SearchCriteriaBuilder $searchCriteriaBuilder
    ) {
        parent::__construct($context);
        $this->productRepository = $productRepository;
        $this->searchCriteriaBuilder = $searchCriteriaBuilder;
    }


    public function execute()
    {
        $this->getResponse()->setHeader('Content-Type', 'text/plain');

        $products = $this->getProductsFromRepository();

        foreach ($products as $product) {
            $this->outputProduct($product);
        }

    }

    /**
     * @return ProductInterface[]
     */
    private function getProductsFromRepository()
    {
        $criteria = $this->searchCriteriaBuilder->create();
        $products = $this->productRepository->getList($criteria);
        return $products->getItems();
    }

    private function outputProduct(ProductInterface $product)
    {
        $this->getResponse()->appendBody(sprintf(
                "%s - %s (%d)\n",
                $product->getName(),
                $product->getSku(),
                $product->getId())
        );
    }
}
```

If there is no output when testing the action in this stage, check the PHP error logs for out-of-memory exceptions.

2. Add a filter so the result list contains only configurable products.

```
public function __construct(
    Context $context,
    ProductRepositoryInterface $productRepository,
    SearchCriteriaBuilder $searchCriteriaBuilder,
    FilterBuilder $filterBuilder
) {
    parent::__construct($context);
    $this->productRepository = $productRepository;
    $this->searchCriteriaBuilder = $searchCriteriaBuilder;
    $this->filterBuilder = $filterBuilder;
}

// ... the execute() method is unchanged ...

/**
 * @return ProductInterface[]
 */
private function getProductsFromRepository()
{
    $this->setProductTypeFilter();

    $criteria = $this->searchCriteriaBuilder->create();
    $products = $this->productRepository->getList($criteria);
    return $products->getItems();
}

private function setProductTypeFilter()
{
    $configProductFilter = $this->filterBuilder
        ->setField('type_id')
        ->setValue(ConfigurableProduct::TYPE_CODE)
        ->setConditionType('eq')
        ->create();
    $this->searchCriteriaBuilder->addFilter([$configProductFilter]);
}
```

Note that the imports have to be adjusted accordingly.

3. Add another filter that is applied using a logical AND operator by adding the following method and calling it from the getProductsFromRepository() method.

```
private function setProductNameFilter()
{
    $nameFilter = $this->filterBuilder
        ->setField('name')
        ->setValue('M%')
        ->setConditionType('like')
        ->create();
```

```
    $this->searchCriteriaBuilder->addFilter([$nameFilter]);
}
```

4. Add a sort order instruction by adding the SortOrderBuilder and SearchCriteriaInterface to the class dependencies and adding the following method (called from the getProductsFromRepository() method).

```
private function setProductOrder()
{
    $sortOrder = $this->sortOrderBuilder
        ->setField('entity_id')
        ->setDirection(SearchCriteriaInterface::SORT_ASC)
        ->create();
    $this->searchCriteriaBuilder->addSortOrder($sortOrder);
}
```

5. Limit the number of product to 6. This can be done by renaming the new method setProductOrder() to setProductPaging() and adding the new 2 lines at the end of the method.

```
private function setProductPaging()
{
    $sortOrder = $this->sortOrderBuilder
        ->setField('entity_id')
        ->setDirection(SearchCriteriaInterface::SORT_ASC)
        ->create();
    $this->searchCriteriaBuilder->addSortOrder($sortOrder);
    $this->searchCriteriaBuilder->setPageSize(6);
    $this->searchCriteriaBuilder->setCurrentPage(1);
}
```

5.4.2. Obtain a list of customers via the customer repository.

- Output the object type.
- Print a list of customers.
- Add a filter to the search criteria.
- Add another filter with a logical OR condition.

Solution

1. Add a new action controller, Customer.php, to the training_repository/repository route created in the previous exercise.
2. In the execute method of the controller, use the customer repository to get a list of customers and print some data.

```
public function execute()
{
    $this->getResponse()->setHeader('content-type', 'text/plain');

    $customers = $this->getCustomersFromRepository();

    $this->getResponse()->appendBody(
        sprintf("List contains %s\n\n", get_class($customers[0]))
```

```
    );
    foreach ($customers as $customer) {
        $this->outputCustomer($customer);
    }
}

/**
 * @return \Magento\Customer\Api\Data\CustomerInterface[]
 */
private function getCustomersFromRepository()
{
    $criteria = $this->searchCriteriaBuilder->create();
    $customers = $this->customerRepository->getList($criteria);
    return $customers->getItems();
}

private function outputCustomer(
    \Magento\Customer\Api\Data\CustomerInterface $customer
) {
    $this->getResponse()->appendBody(sprintf(
        "\"%s %s\" <%s> (%s)\n",
        $customer->getFirstname(),
        $customer->getLastname(),
        $customer->getEmail(),
        $customer->getId()
    ));
}
```

3. Output the type of the objects returned by the repository.

```
public function execute()
{
    $this->getResponse()->setHeader('content-type', 'text/plain');

    $customers = $this->getCustomersFromRepository();

    $this->getResponse()->appendBody(
        sprintf("List contains %s\n\n", get_class($customers[0]))
    );
    foreach ($customers as $customer) {
        $this->outputCustomer($customer);
    }
}
```

4. Add two filters with a logical OR condition by specifying them as a filter group.

```
namespace Training\Repository\Controller\Repository;

use Magento\Customer\Api\CustomerRepositoryInterface
use Magento\Framework\Api\FilterBuilder;
use Magento\Framework\Api\Search\FilterGroupBuilder;
use Magento\Framework\Api\SearchCriteriaBuilder;
use Magento\Framework\App\Action\Action;
```

```php
use Magento\Framework\App\Action\Context;

class Customer extends Action
{
    /**
     * @var CustomerRepositoryInterface
     */
    private $customerRepository;

    /**
     * @var SearchCriteriaBuilder
     */
    private $searchCriteriaBuilder;

    /**
     * @var FilterGroupBuilder
     */
    private $filterGroupBuilder;

    /**
     * @var FilterBuilder
     */
    private $filterBuilder;

    public function __construct(
        Context $context,
        CustomerRepositoryInterface $customerRepository,
        SearchCriteriaBuilder $searchCriteriaBuilder,
        FilterGroupBuilder $filterGroupBuilder,
        FilterBuilder $filterBuilder
    ) {
        parent::__construct($context);
        $this->customerRepository = $customerRepository;
        $this->searchCriteriaBuilder = $searchCriteriaBuilder;
        $this->filterGroupBuilder = $filterGroupBuilder;
        $this->filterBuilder = $filterBuilder;
    }

    public function execute()
    {
        $this->getResponse()->setHeader('content-type', 'text/plain');

        $this->addEmailFilter();
        $this->addNameFilter();
        $customers = $this->getCustomersFromRepository();

        $this->getResponse()->appendBody(
            sprintf("List contains %s\n\n", get_class($customers[0]))
        );

        foreach ($customers as $customer) {
            $this->outputCustomer($customer);
        }
    }
```

```
private function addEmailFilter()
    {
        $emailFilter = $this->filterBuilder
            ->setField('email')
            ->setValue('%@dmail.com')
            ->setConditionType('like')
            ->create();
        $this->filterGroupBuilder->addFilter($emailFilter);
    }

    private function addNameFilter()
    {
        $nameFilter = $this->filterBuilder
            ->setField('firstname')
            ->setValue('Hans')
            ->setConditionType('eq')
            ->create();
        $this->filterGroupBuilder->addFilter($nameFilter);
    }

    /**
     * @return \Magento\Customer\Api\Data\CustomerInterface[]
     */
    private function getCustomersFromRepository()
    {
        $this->searchCriteriaBuilder->setFilterGroups(
            [$this->filterGroupBuilder->create()]
        );
        $criteria = $this->searchCriteriaBuilder->create();
        $customers = $this->customerRepository->getList($criteria);
        return $customers->getItems();
    }

    private function outputCustomer(
        \Magento\Customer\Api\Data\CustomerInterface $customer
    ) {
        $this->getResponse()->appendBody(sprintf(
            "\"%s %s\" <%s> (%s)\n",
            $customer->getFirstname(),
            $customer->getLastname(),
            $customer->getEmail(),
            $customer->getId()
        ));
    }
}
```

5.4.3. Create a service API and repository for a custom entity.

- Try to follow best practices.
- The custom example entity should use a flat table for storage.
- The repository only needs to contain a `getList()` method.

Solution

1. Create a new flat table entity called Example with a model, resource model, and collection. Refer to the ORM unit of the training or the exercise solution code archive for details. The model, resource model, and collection code is boilerplate and is not included in this document.

```
Repository
├── Model
│   ├── Example.php
│   └── Resource
│       ├── Example
│       │   └── Collection.php
│       └── Example.php
└── Setup
    ├── InstallSchema.php
    └── UpgradeData.php
```

2. Use a module install class to create the matching table, and a module data upgrade class to create a couple of example records in the table.

```php
<?php

namespace Training\Repository\Setup;

use Magento\Framework\DB\Adapter\AdapterInterface;
use Magento\Framework\DB\Ddl\Table as DdlTable;
use Magento\Framework\Setup\InstallSchemaInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\SchemaSetupInterface;

class InstallSchema implements InstallSchemaInterface
{
    public function install(
        SchemaSetupInterface $setup,
        ModuleContextInterface $context
    ) {
        $setup->startSetup();
        $tableName = $setup->getTable('training_repository_example');
        $ddlTable = $setup->getConnection()->newTable(
            $tableName
        );
        $ddlTable->addColumn(
            'example_id',
            DdlTable::TYPE_INTEGER,
            null,
            [
                'identity' => true,
                'unsigned' => true,
                'nullable' => false,
                'primary' => true
            ]
        )->addColumn(
```

```
            'name',
            DdlTable::TYPE_TEXT,
            255,
            ['nullable' => false]
        )->addColumn(
            'created_at',
            DdlTable::TYPE_TIMESTAMP,
            null,
            ['nullable' => false, 'default' => DdlTable::TIMESTAMP_INIT]
        )->addColumn(
            'updated_at',
            DdlTable::TYPE_TIMESTAMP,
            null,
            ['nullable' => false, 'default' => DdlTable::TIMESTAMP_INIT]
        )->addIndex(
            $setup->getIdxName(
                $tableName,
                ['name'],
                AdapterInterface::INDEX_TYPE_UNIQUE
            ),
            ['name'],
            ['type' => AdapterInterface::INDEX_TYPE_UNIQUE]
        );

        $setup->getConnection()->createTable($ddlTable);

        $setup->endSetup();
    }
}
```

```
<?php

namespace Training\Repository\Setup;

use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\ModuleDataSetupInterface;
use Magento\Framework\Setup\UpgradeDataInterface;

class UpgradeData implements UpgradeDataInterface
{
    public function upgrade(
        ModuleDataSetupInterface $setup,
        ModuleContextInterface $context
    ) {
        $dbVersion = $context->getVersion();

        if (version_compare($dbVersion, '0.1.1', '<')) {
            $tableName = $setup->getTable('training_repository_example');
            $setup->getConnection()->insertMultiple(
                $tableName,
                [
                    ['name' => 'Foo'],
                    ['name' => 'Bar'],
                    ['name' => 'Baz'],
                    ['name' => 'Qux'],
```

```
                ]
            );
        }
    }
}
```

3. Add an interface Training\Repository\Api\ExampleRepositoryInterface. It contains only the getList() method. No framework interface needs to be extended.

```
<?php

namespace Training\Repository\Api;

use Magento\Framework\Api\SearchCriteriaInterface;

interface ExampleRepositoryInterface
{
    /**
     * @return Data\ExampleSearchResultsInterface
     */
    public function getList(SearchCriteriaInterface $searchCriteria);
}
```

4. Add an interface for the API data model Training\Repository\Api\Data\ExampleInterface with getters and setters for all the properties that should be accessible from the outside.

```
<?php

namespace Training\Repository\Api\Data;

interface ExampleInterface
{
    /**
     * @param int $id
     * @return $this
     */
    public function setId($id);

    /**
     * @return int
     */
    public function getId();

    /**
     * @return string
     */
```

```
public function getName();

    /**
     * @param string $name
     * @return $this
     */
    public function setName($name);

    /**
     * @return string
     */
    public function getCreatedAt();

    /**
     * @param string $createdAt
     * @return $this
     */
    public function setCreatedAt($createdAt);

    /**
     * @return string
     */
    public function getModifiedAt();

    /**
     * @param string $modifiedAt
     * @return $this
     */
    public function setModifiedAt($modifiedAt);
}
```

5. Add an interface Training\Repository\Api\Data\ExampleSearchResultsInterface that extends Magento\Framework\Api\SearchResultsInterface.

   It can inherit all methods, or specify getItems() and setItems() to provide more specific phpdoc type hints.

```
<?php

namespace Training\Repository\Api\Data;

interface ExampleSearchResultsInterface
    extends \Magento\Framework\Api\SearchResultsInterface
{
    /**
     * @api
     * @return \Training\Repository\Api\Data\ExampleInterface[]
     */
```

```
public function getItems();

    /**
     * @api
     * @param \Training\Repository\Api\Data\ExampleInterface[] $items
     * @return $this
     */
    public function setItems(array $items = null);
}
```

6. Specify the preferences configuration for these three new interfaces in an etc/di.xml file.

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../../lib/internal/Magento/Framework/ObjectManager/etc/config.xsd">
    <preference for="Training\Repository\Api\ExampleRepositoryInterface"
                type="Training\Repository\Model\ExampleRepository"/>
    <preference for="Training\Repository\Api\Data\ExampleInterface"
                type="Training\Repository\Model\Example"/>
    <preference for="Training\Repository\Api\Data\ExampleSearchResultsInterface"
                type="Magento\Framework\Api\SearchResults"/>
</config>
```

7. Make the Training\Repository\Model\Example class implement the Api\Data\ExampleInterface.

```
<?php

namespace Training\Repository\Model;

use Magento\Framework\Model\AbstractModel;
use Training\Repository\Api\Data\ExampleInterface;

class Example extends AbstractModel implements ExampleInterface
{
    protected function _construct()
    {
        $this->_init(Resource\Example::class);
    }

    public function getName()
    {
        return $this->_getData('name');
    }
```

```
public function setName($name)
    {
        $this->setData('name', $name);
    }

    public function getCreatedAt()
    {
        return $this->_getData('created_at');
    }

    public function setCreatedAt($createdAt)
    {
        $this->setData('modified_at', $createdAt);
    }

    public function getModifiedAt()
    {
        return $this->_getData('modified_at');
    }

    public function setModifiedAt($modifiedAt)
    {
        $this->setData('modified_at', $modifiedAt);
    }
}
```

8. Finally it is time to create the repository implementation.

   The getList() method creates an example collection instance and applies the SearchCriteria using the appropriate methods on the collection.

   Then the collection is loaded, and all entities are converted into the configured object implementation for the Api\Data\ExampleInterface.

   For this implementation it means that the conversion happens to the same instance, but should the DI configuration for the interface change, this ensures that change will take effect.

```
<?php

namespace Training\Repository\Model;

use Magento\Framework\Api\Search\FilterGroup;
use Magento\Framework\Api\SearchCriteriaInterface;
use Training\Repository\Api\Data\ExampleInterface;
use Training\Repository\Api\Data\ExampleInterfaceFactory as ExampleDataFactory;
use Training\Repository\Api\Data\ExampleSearchResultsInterface;
use Training\Repository\Api\Data\ExampleSearchResultsInterfaceFactory;
use Training\Repository\Api\ExampleRepositoryInterface;
use Training\Repository\Model\Example as ExampleModel;
use Training\Repository\Model\Resource\Example\Collection as ExampleCollection;
```

```php
class ExampleRepository implements ExampleRepositoryInterface
{
    /**
     * @var ExampleSearchResultsInterfaceFactory
     */
    private $searchResultsFactory;

    /**
     * @var ExampleFactory
     */
    private $exampleFactory;

    /**
     * @var ExampleDataFactory
     */
    private $exampleDataFactory;

    public function __construct(
        ExampleSearchResultsInterfaceFactory $searchResultsFactory,
        ExampleFactory $exampleFactory,
        ExampleDataFactory $exampleDataFactory
    ) {
        $this->searchResultsFactory = $searchResultsFactory;
        $this->exampleFactory = $exampleFactory;
        $this->exampleDataFactory = $exampleDataFactory;
    }

    /**
     * @return ExampleSearchResultsInterface
     */
    public function getList(SearchCriteriaInterface $searchCriteria)
    {
        /** @var ExampleCollection $collection */
        $collection = $this->exampleFactory->create()->getCollection();

        /** @var ExampleSearchResultsInterface $searchResults */
        $searchResults = $this->searchResultsFactory->create();
        $searchResults->setSearchCriteria($searchCriteria);

        $this->applySearchCriteriaToCollection($searchCriteria, $collection);
        $examples = $this->convertCollectionToDataItemsArray($collection);

        $searchResults->setTotalCount($collection->getSize());
        $searchResults->setItems($examples);

        return $searchResults;
    }
```

```
    private function addFilterGroupToCollection(
        FilterGroup $filterGroup,
        ExampleCollection $collection
    ) {
        $fields = [];
        $conditions = [];
        foreach ($filterGroup->getFilters() as $filter) {
            $condition = $filter->getConditionType() ?
                $filter->getConditionType() :
                'eq';
            $fields[] = $filter->getField();
            $conditions[] = [$condition => $filter->getValue()];
        }
        if ($fields) {
            $collection->addFieldToFilter($fields, $conditions);
        }
    }
    private function convertCollectionToDataItemsArray(
        ExampleCollection $collection
    ) {
        $examples = array_map(function (ExampleModel $example) {
            /** @var ExampleInterface $dataObject */
            $dataObject = $this->exampleDataFactory->create();
            $dataObject->setId($example->getId());
            $dataObject->setName($example->getName());
            $dataObject->setCreatedAt($example->getCreatedAt());
            $dataObject->setModifiedAt($example->getModifiedAt());
            return $dataObject;
        }, $collection->getItems());
        return $examples;
    }
    private function applySearchCriteriaToCollection(
        SearchCriteriaInterface $searchCriteria,
        ExampleCollection $collection
    ) {
        $this->applySearchCriteriaFiltersToCollection(
            $searchCriteria,
            $collection
        );
        $this->applySearchCriteriaSortOrdersToCollection(
            $searchCriteria,
            $collection
        );
        $this->applySearchCriteriaPagingToCollection(
            $searchCriteria,
            $collection
        );
    }
```

```
private function applySearchCriteriaFiltersToCollection(
        SearchCriteriaInterface $searchCriteria,
        ExampleCollection $collection
    ) {
        foreach ($searchCriteria->getFilterGroups() as $group) {
            $this->addFilterGroupToCollection($group, $collection);
        }
    }

    private function applySearchCriteriaSortOrdersToCollection(
        SearchCriteriaInterface $searchCriteria,
        ExampleCollection $collection
    ) {
        $sortOrders = $searchCriteria->getSortOrders();
        if ($sortOrders) {
            $isAscending =
                $sortOrder->getDirection() == SearchCriteriaInterface::SORT_ASC;
            foreach ($sortOrders as $sortOrder) {
                $collection->addOrder(
                    $sortOrder->getField(),
                    $isAscending ? 'ASC' : 'DESC'
                );
            }
        }
    }

    private function applySearchCriteriaPagingToCollection(
        SearchCriteriaInterface $searchCriteria,
        ExampleCollection $collection
    ) {
        $collection->setCurPage($searchCriteria->getCurrentPage());
        $collection->setPageSize($searchCriteria->getPageSize());
    }
}
```

9. Create an action controller to test the result.

```
<?php

namespace Training\Repository\Controller\Repository;

use Magento\Framework\Api\FilterBuilder;
use Magento\Framework\Api\SearchCriteriaBuilder;
use Magento\Framework\App\Action\Action;
use Magento\Framework\App\Action\Context;
use Training\Repository\Api\ExampleRepositoryInterface;
```

```
class Example extends Action
{
    /**
     * @var ExampleRepositoryInterface
     */
    private $exampleRepository;

    /**
     * @var SearchCriteriaBuilder
     */
    private $searchCriteriaBuilder;

    /**
     * @var FilterBuilder
     */
    private $filterBuilder;

    public function __construct(
        Context $context,
        ExampleRepositoryInterface $exampleRepository,
        SearchCriteriaBuilder $searchCriteriaBuilder,
        FilterBuilder $filterBuilder
    ) {
        $this->exampleRepository = $exampleRepository;
        $this->searchCriteriaBuilder = $searchCriteriaBuilder;
        $this->filterBuilder = $filterBuilder;
        parent::__construct($context);
    }

    public function execute()
    {
        $this->getResponse()->setHeader('content-type', 'text/plain');

        $filters = array_map(function ($name) {
            return $this->filterBuilder
                ->setConditionType('eq')
                ->setField('name')
                ->setValue($name)
                ->create();
        }, ['Foo', 'Bar', 'Baz', 'Qux']);
        $this->searchCriteriaBuilder->addFilter($filters);
        $examples = $this->exampleRepository->getList(
            $this->searchCriteriaBuilder->create()
        )->getItems();
```

```
foreach ($examples as $example) {
            $this->getResponse()->appendBody(sprintf(
                "%s (%d)\n",
                $example->getName(),
                $example->getId()
            ));
        }

    }
}
```

Module 5. Data API

5.5.1. Create a new entity `category_countries`.

- Include `category_country_id`, `category_id`, `country_id`.
- Add a few records to that table (using `DataInstallScript`).
- Add an extension attribute "`countries`" to the category.

Solution

You will be provided with a code archive containing the solutions for the exercises in this module.

Module 6. Web API

5.6.1. Create scripts that make SOAP calls.

- Create a `php-script` that performs a SOAP call to the customer repository `getById()` method.
- Create a `php-script` that performs a SOAP call to the customer repository `getList()` method. Define the filter & sorting options in the `SearchCriteria` parameter.
- Create a `php-script` that performs a SOAP call to the catalog product repository `getList()` method.
- Add a new attribute in the Admin and make a SOAP call to the catalog product repository `get()` method to obtain a product with a list of attributes. Make sure your new attribute is there.

Solution

You will be provided with a code archive containing the solutions for the exercises in this module.

5.6.2. Perform an API call to the "`V1/customers/1`" path.

- Explore the `Magento_Customer` module and find other examples of the available services. Perform a call to some service you've found there.

Solution

```php
<?php

require('vendor/zendframework/zend-server/Zend/Server/Client.php');
require('vendor/zendframework/zend-soap/Zend/Soap/Client.php');
require('vendor/zendframework/zend-soap/Zend/Soap/Client/Common.php');

$wsdlUrl = 'http://magento.loc/m2-0.74/soap/V1/?wsdl&services=customerAccountManagementV1,
                                                customerCustomerRepositoryV1';
$token  = '12345';

$opts = ['http' => ['header' => "Authorization: Bearer " . $token]];
$context = stream_context_create($opts);

$soapClient = new \Zend\Soap\Client($wsdlUrl);
$soapClient->setSoapVersion(SOAP_1_2);
$soapClient->setStreamContext($context);
$result = $soapClient->customerAccountManagementV1IsReadonly(array('customerId' => 1));
var_dump($result);
```

5.6.3. Create your own Data API class and make it available through the Web API.

- Make it anonymous and test how it works through the REST.

Solution

1. In order to perform a REST API call to the Customer module, perform the following two http calls:

   a. `curl -X POST "http://magento.loc/m2-0.74/index.php/rest/V1/integration/admin/token" -H "Content-Type:application/json" -d '{"username":"_ADMIN_USERNAME_", "password":"_ADMIN_PASSWORD_"}'`

      It will return a token.

   b. `curl -X GET "http://magento.loc/m2-0.74/index.php/rest/V1/customers/1" -H "Authorization: Bearer _ADMIN_TOKEN_"`

      Where _ADMIN_TOKEN_ is the one from the previous call.

2. Create a module Api:

module.xml

```
<?xml version="1.0"?>
<!--
/**
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../../lib/internal/Magento/Framework\
/Module/etc/module.xsd">
    <module name="Training_Api" setup_version="1.0.0">
    </module>
</config>
```

di.xml

```
<?xml version="1.0"?>
<!--
/**
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../../lib/internal/Magento/Framework\
/ObjectManager/etc/config.xsd">
    <preference for="Training\Api\Api\Data\HelloInterface" type="Training\Api\Model\Hello" />
</config>
```

webapi.xml

```
<?xml version="1.0"?>
<!--
/**
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="../../../../../app/code/Magento/Webapi/etc/webapi.xsd">

    <route url="/V1/traininghello" method="GET">
       <service class="Training\Api\Api\Data\HelloInterface" method="sayHello"/>
       <resources>
            <resource ref="anonymous" />
       </resources>
    </route>
</routes>
```

Api/Api/Data/HelloInterface.php

```
<?php

namespace Training\Api\Api\Data;

interface HelloInterface {

    /**
     * Hello method
     *
     * @return string|null
     */
    public function sayHello();
}
```

Api/Model/Hello.php

```
<?php

namespace Training\Api\Model;

use Training\Api\Api\Data;

class Hello implements \Training\Api\Api\Data\HelloInterface {

public function sayHello() {
        return "HELLO WORLD!";
    }
}
```

Call it by hitting a URL: http://magento.loc/m2-0.74/index.php/rest/V1/traininghello

You can do this either by using curl or using a browser.

The result should be:

```
<?xml version="1.0"?>
<response>HELLO WORLD!</response>
```

Unit Six. AdminHTML

Module 3. Grids: Part 1

6.3.1. Configurable Products

- Add a new filter for configurable products that filters by number of configurable options.
- It should be available on the products grid, as a dropdown with a list of possible values (according to which configurables are in the database).
- When selected, it should show a list of configurable products with the appropriate options.

Solution

You will be provided with a code archive containing the solutions for the exercises in this module.

6.3.2. Add a New Attribute

- Add new attribute to the product, of type varchar (call it `product_series`).
- Create a filter in the product grid for that attribute.

Solution

You will be provided with a code archive containing the solutions for the exercises in this module.

Module 4. Grids: Part 2

6.4.1. Add a New Column

- Add new column to the order table `require_verification.`
- It should always equal “1” if an order is placed through the checkout.
- It should equal “0” if it was placed in the Admin.

Solution

You will be provided with a code archive containing the solutions for the exercises in this module.

6.4.2. Add a New Attribute

- Add new column to the order table `require_verification`.
- It should always be set to “1” when an order is created.
- Do this by setting the “default” property of a corresponding column.
- Note that, in general, it will require a more sophisticated customization of a `Magento/sales` module.

Solution

You will be provided with a code archive containing the solutions for the exercises in this module.

6.4.3. Mass Action

- For the `require_verification` field, add a mass action to the orders grid which changes it to "0" for all selected orders.

Solution

You will be provided with a code archive containing the solutions for the exercises in this module.

6.4.4. Create a Table

- Create a new table, `computer_games`, with the following fields:

1. `game_id`
2. `name` (varchar)
3. `type` (RPG)
4. `RTS`
5. `MMO`
6. `Simulator`
7. `Shooter`
8. `trial_period` (period in days)
9. `release_date`

- For this table, create a grid with columns that correspond to the fields.
- Make `release_date` column optional (not visible by default).
- Create filters that correspond to fields (text for `name`, dropdown for `type`, from-to for `trial_period`, date for `release_date`).

Solution

You will be provided with a code archive containing the solutions for the exercises in this module.

Module 5. Adminhtml Forms

6.5.1. Add a Form

- For the `computer_games` table and grid created in the previous grid section, add a form.
- Use following field types:
  - `name` (text)
  - `type` (dropdown: RPG, RTS, MMO, Simulator, Shooter)
  - `trial_period` (integer)
  - `release_date` (date)
- Make sure the `save/delete/back` buttons are available and functioning.

Solution

You will be provided with a code archive containing the solutions for the exercises in this module.

Module 6. Adminhtml: System Configuration – Menu – ACL

6.6.1. System Configuration

- Create a new element in the system configuration.
    - Name it “test”.
    - Put it into the General section.
    - Make it a “yes/no” select.
- Create a new element in the system configuration with custom code that renders “Hello World”.

Solution

You will be provided with a code archive containing the solutions for the exercises in this module.

6.6.2. Menu

Create a submenu in the Catalog/Product menu called “Games”. It should lead to the games grid created earlier.

Solution

You will be provided with a code archive containing the solutions for the exercises in this module.

6.6.3. ACL

- Create a new page in Admin that renders “Hello World”.
- Create a new role for this page.
- Create a new user and assign the user access to the page. (Verify that the user does have access.)

Solution

You will be provided with a code archive containing the solutions for the exercises in this module.

