# Contents

# Unit 1. Preparation & Configuration

## Module 3. Magento 2 Overview

### 1.3.1. (module name: Unit1_Test and Unit1_Test2) 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/Unit1/Test.
Create a file app/code/Unit1/Test/etc/module.xml:

```xml
<?xml version="1.0"?>
<!--
/**
 *
 * Copyright © 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="Unit1_Test" setup_version="0.0.1"></module>
</config>
```

2. Register your module with app/code/Unit1/Test/registration.php

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'Unit1_Test',
    __DIR__
);
```

3. Enable your module. Run `php bin/magento module:enable Unit1_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 php bin/magento cache:clean and php bin/magento cache:flush) 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` and php bin/magento cache:flush

6. Create a folder app/code/Unit1/Test2 and file app/code/Unit1/Test2/etc/module.xml:

```xml
<?xml version="1.0"?>
<!--
/**
 *
 * Copyright © 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="Unit1_Test2" setup_version="0.0.1">
        <sequence>
            <module name="Unit1_Test"/>
        </sequence>
    </module>
</config>
```

7. Enable your module. Run php bin/magento module:enable Unit1_Test2

8. Clean the cache, and test whether your module is working.

9. You can disable Unit1_Test by setting its value to 0 in the etc/config.php or run php bin/magento module:disable Unit1_Test1

# Module 5. Development Operations

## 1.5.1. Mode

- Make sure the mode is set to Developer (for example, uncomment  SetEnv MAGE_MODE default in the .htaccess file)
- Then, go into the lib\internal\Magento\Framework\App\Bootstrap.php (or vendor\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 or vendor\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. (module name: Unit1_Test and Unit1_Test2) Object Manager

Go back to the two modules created you created in Exercise 1.3.1, `Unit1_Test` and `Unit1_Test2`*.
In `Unit1_Test,` 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:**

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit1\Test\MagentoU;

class Test
{
    protected $justAParameter;

    protected $data;

    protected $unit1ProductRepository;

    public function __construct(
        \Magento\Catalog\Api\ProductRepositoryInterface $productRepository,
        \Magento\Catalog\Model\ProductFactory $productFactory,
        \Magento\Checkout\Model\Session $session,
        \Unit1\Test\Api\ProductRepositoryInterface $unit1ProductRepository,
        $justAParameter = false,
        array $data = []
    ) {
        $this->justAParameter = $justAParameter;
        $this->data = $data;
        $this->unit1ProductRepository = $unit1ProductRepository;
    }
}
```

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

Now, just only for learning purposes, add interface for Magento\Catalog\Model\ProductRepository, as below:

```php
class ProductRepository implements \Magento\Catalog\Api\ProductRepositoryInterface,
    \Unit1\Test\Api\ProductRepositoryInterface
```

Note: Remove it after checking the result.

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

```xml
<?xml version="1.0"?>
<!--
/**
 *
 * Copyright © 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:ObjectManager/etc/config.xsd">
    <preference for="Unit1\Test\Api\ProductRepositoryInterface"
type="Magento\Catalog\Model\ProductRepository" />
</config>
```

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

Next, modify the parameters of a constructor:
```xml
<type name="Unit1\Test\MagentoU\Test">
    <arguments>
        <argument name="justAParameter" xsi:type="string">Hello world!</argument>
        <argument name="data" xsi:type="array">
            <item name="test-array-item" xsi:type="string">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 `Test2` and add another item to the `$data` array from there.

# Module 7. Plugins

## 1.7.1. * Optional Exercise: 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, examine a couple of interceptors and note how similar they are to each other – for example, `generated/code/Magento/Catalog/Model/Product/Interceptor.php`.

## 1.7.2. (module name: Unit1_Plugins) Plugins 2

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

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

**Solution**

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 . "(!)"`

For this task, make it happen twice with `After` and `Around` plugins.

**1.** Create Plugins di config records:
```xml
<!--
/**
 *
 * Copyright © 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:ObjectManager/etc/config.xsd">

    <type name="Magento\Catalog\Model\Product">
        <plugin name="afterPricePlugin" type="Unit1\Plugins\Plugin\AfterPricePlugin"
sortOrder="1" disabled="false" />
    </type>

    <type name="Magento\Theme\Block\Html\Footer">
        <plugin name="footerPlugin" type="Unit1\Plugins\Plugin\AfterFooterPlugin"
sortOrder="2" disabled="false" />
    </type>

    <type name="Magento\Theme\Block\Html\Breadcrumbs">
        <plugin name="aroundBreadcrumbsPlugin"
type="Unit1\Plugins\Plugin\AroundBreadcrumbsPlugin" sortOrder="3" disabled="false" />
    </type>

    <type name="Magento\Theme\Block\Html\Breadcrumbs">
        <plugin name="beforeBreadcrumbsPlugin"
type="Unit1\Plugins\Plugin\BeforeBreadcrumbsPlugin" sortOrder="4" disabled="false" />
    </type>

</config>
```

2. Create AfterFooterPlugin:
```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit1\Plugins\Plugin;
```

```php
/**
 * Class AfterFooterPlugin
 * @package Unit1\Plugins\Plugin
 */
class AfterFooterPlugin
{
    /**
     * @param \Magento\Theme\Block\Html\Footer $subject
     * @param $result
     * @return string
     */
    public function afterGetCopyright(\Magento\Theme\Block\Html\Footer $subject, $result)
    {
        return 'Customized copyright!';
    }
}
```

3. Create AfterPricePlugin:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit1\Plugins\Plugin;

/**
 * Class AfterPricePlugin
 * @package Unit1\Plugins\Plugin
 */
class AfterPricePlugin
{
    /**
     * @param \Magento\Catalog\Model\Product $subject
     * @param $result
     * @return mixed
     */
    public function afterGetPrice(\Magento\Catalog\Model\Product $subject, $result)
    {
        return $result + 0.5;
    }
}
```

4. Create AroundBreadcrumbsPlugin:

```php
<?php
/**
 *
 * Copyright © 2019 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit1\Plugins\Plugin;

/**
 * Class AroundBreadcrumbsPlugin
 * @package Unit1\Plugins\Plugin
 */
class AroundBreadcrumbsPlugin
```

```
    {
        /**
         * @param \Magento\Theme\Block\Html\Breadcrumbs $subject
         * @param callable $proceed
         * @param $crumbName
         * @param $crumbInfo
         */
        public function aroundAddCrumb(
            \Magento\Theme\Block\Html\Breadcrumbs $subject, callable $proceed,
            $crumbName, $crumbInfo
        )
        {
            $crumbInfo['label'] = $crumbInfo['label'].'(!a)';
            $proceed($crumbName, $crumbInfo);
        }
    }
```

5.  Create BeforeBreadcrumbsPlugin:

```php
<?php
/**
 *
 * Copyright © 2019 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit1\Plugins\Plugin;

/**
 * Class BeforeBreadcrumbsPlugin
 * @package Unit1\Plugins\Plugin
 */
class BeforeBreadcrumbsPlugin
{
    /**
     * @param \Magento\Theme\Block\Html\Breadcrumbs $subject
     * @param $crumbName
     * @param $crumbInfo
     * @return array
     */
    public function beforeAddCrumb(\Magento\Theme\Block\Html\Breadcrumbs $subject,
$crumbName, $crumbInfo)
    {
        $crumbInfo['label'] = $crumbInfo['label'].'(!b)';
        return [$crumbName, $crumbInfo];
    }
}
```

6.  Clean cache refresh home page. Check home page product prices as well as footer and breadcrumbs are worked well.

## Module 8. Events

**1.8.1. (module name Unit1_LogPathInfo) 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
<?xml version="1.0"?>
<!--
/**
 *
 * Copyright © 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="unit1_test"
                  instance="Unit1\LogPathInfo\Observer\Log" shared="false" />
    </event>
</config>
```

2.  Create an Observer:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit1\LogPathInfo\Observer;

/**
 * Class Log
 * @package Unit1\LogPathInfo\Observer
 */
class Log implements \Magento\Framework\Event\ObserverInterface
{
    /**
     * @var \Psr\Log\LoggerInterface
     */
    private $_logger;
    /**
     * @var \Magento\Framework\App\RequestInterface
     */
    private $_request;

    /**
```

```
        * Log constructor.
        * @param \Psr\Log\LoggerInterface $logger
        * @param \Magento\Framework\App\RequestInterface $request
        */
      public function __construct(
          \Psr\Log\LoggerInterface $logger,
          \Magento\Framework\App\RequestInterface $request)
      {
          $this->_logger = $logger;
          $this->_request = $request;
      }

      /**
        * @param \Magento\Framework\Event\Observer $observer
        */
      public function execute(\Magento\Framework\Event\Observer $observer) {
          $this->_logger->critical(
              'Request URI: ' . $this->_request->getPathInfo()
          );
      }
}
```

**Result:** Now we know how log some data to the file. You can see the result in the var/log/system.log file.

# Module 9. Module Configuration

## 1.9.1. (module name: Unit1_CustomConfig) 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 `custom_config.xml` and `custom_config.xsd` files.
**Phase 2**: Create PHP files to process them: `Config`, `Convertor`, `Reader` and `SchemaLocator`.create virtually (using di.xml).
**Phase 3**: Test: In this example we will create a new page.

**Solution**
Let's follow through each step.

### Phase 1

**1.1)**    **Create `etc/custom_config.xml`:**

```
<?xml version="1.0"?>
<!--
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<config xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation="urn:magento:module:Unit1_CustomConfig:etc/custom_confi
```

```
g.xsd">
      <welcome_message store_id="1">Welcome USA customer!</welcome_message>
      <welcome_message store_id="2">Welcome Canadian customer!</welcome_message>
      <welcome_message store_id="3">Welcome EU customer!</welcome_message>
</config>
```

**1.2)    Create `etc/custom_config.xsd`:**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="config" type="configType">
     <xs:annotation>
       <xs:documentation>/**
 * ACL. Can be queried for relations between roles and resources.
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */</xs:documentation>
     </xs:annotation>
  </xs:element>
  <xs:complexType name="welcome_messageType">
     <xs:simpleContent>
       <xs:extension base="xs:string">
         <xs:attribute type="xs:string" name="store_id" use="optional"/>
       </xs:extension>
     </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="configType">
     <xs:sequence>
       <xs:element type="welcome_messageType" name="welcome_message"
maxOccurs="unbounded" minOccurs="0"/>
     </xs:sequence>
  </xs:complexType>
</xs:schema>
```

## Phase 2

**2.1) Create Config class:**

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

```
namespace Unit1\CustomConfig\Model;

use Magento\Framework\Config\CacheInterface;
use Magento\Framework\Config\ReaderInterface;

/**
 * Class Config
 * @package Unit1\CustomConfig\Model
 */
class Config extends \Magento\Framework\Config\Data
{
    /**
     * Config constructor.
     * @param Config\Reader $reader
     * @param CacheInterface $cache
     * @param string $cacheId
     */
    public function __construct(ReaderInterface $reader, CacheInterface $cache, $cacheId =
'')
    {
        parent::__construct($reader, $cache, $cacheId);
    }
}
```

**2.2) Create Reader virtual class (etc/di.xml):**

```xml
<?xml version="1.0"?>
<!--
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
…
<!--Config Reader Model-->
<virtualType name="Unit1\CustomConfig\Model\Config\Reader"
type="Magento\Framework\Config\Reader\Filesystem">
    <arguments>
        <argument name="fileName" xsi:type="string">custom_config.xml</argument>
        <argument name="defaultScope" xsi:type="string">global</argument>
        <argument name="schemaLocator"
xsi:type="object">Unit1\CustomConfig\Model\Config\SchemaLocator</argument>
        <argument name="converter"
xsi:type="object">Unit1\CustomConfig\Model\Config\Converter</argument>
        <argument name="idAttributes" xsi:type="array">
            <item name="/config/welcome_message" xsi:type="string">store_id</item>
        </argument>
    </arguments>
</virtualType>
…
```

**2.3) Create schemaLocator virtual class (etc/di.xml):**

```xml
<!--Config Schema Locator class    -->
<virtualType name="Unit1\CustomConfig\Model\Config\SchemaLocator"
type="Magento\Framework\Config\GenericSchemaLocator">
    <arguments>
        <argument name="schema" xsi:type="string">custom_config.xsd</argument>
        <argument name="perFileSchema" xsi:type="string">custom_config.xsd</argument>
        <argument name="moduleName" xsi:type="string">Unit1_CustomConfig</argument>
    </arguments>
</virtualType>
```

**2.4) Create converter class:**

```php
<?php
/**
 * Copyright © Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit1\CustomConfig\Model\Config;

class Converter implements \Magento\Framework\Config\ConverterInterface
{
    /**
     * Convert dom node tree to array
     *
     * @param \DOMDocument $source
     * @return array
     * @throws \InvalidArgumentException
     *
     * @SuppressWarnings(PHPMD.CyclomaticComplexity)
     */
    public function convert($source)
    {
        $output = [];
        $xpath = new \DOMXPath($source);
        $messages = $xpath->evaluate('/config/welcome_message');
        /** @var $messageNode \DOMNode */
        foreach ($messages as $messageNode) {
            $storeId = $this->_getAttributeValue($messageNode, 'store_id');

            $data = [];
            /** @var $childNode \DOMNode */
            foreach ($messageNode->childNodes as $childNode) {
                $data = ['message' => $childNode->nodeValue];
            }
            $output['messages'][$storeId] = $data;
        }

        return $output;
    }

    /**
     * Get attribute value
     *
     * @param \DOMNode $input
```

```
     * @param string $attributeName
     * @param string|null $default
     * @return null|string
     */
    protected function _getAttributeValue(\DOMNode $input, $attributeName, $default = null)
    {
        $node = $input->attributes->getNamedItem($attributeName);
        return $node ? $node->nodeValue : $default;
    }
}
```

**di.xml:**

```xml
<?xml version="1.0"?>
<!--
/**
 *
 * Copyright © 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:ObjectManager/etc/config.xsd">
    <!--Use the power of DI!-->

    <!--Config Schema Locator class    -->
    <virtualType name="Unit1\CustomConfig\Model\Config\SchemaLocator"
type="Magento\Framework\Config\GenericSchemaLocator">
        <arguments>
            <argument name="schema" xsi:type="string">custom_config.xsd</argument>
            <argument name="perFileSchema" xsi:type="string">custom_config.xsd</argument>
            <argument name="moduleName" xsi:type="string">Unit1_CustomConfig</argument>
        </arguments>
    </virtualType>
    <!--Config Reader Model-->
    <virtualType name="Unit1\CustomConfig\Model\Config\Reader"
type="Magento\Framework\Config\Reader\Filesystem">
        <arguments>
            <argument name="fileName" xsi:type="string">custom_config.xml</argument>
            <argument name="defaultScope" xsi:type="string">global</argument>
            <argument name="schemaLocator"
xsi:type="object">Unit1\CustomConfig\Model\Config\SchemaLocator</argument>
            <argument name="converter"
xsi:type="object">Unit1\CustomConfig\Model\Config\Converter</argument>
            <argument name="idAttributes" xsi:type="array">
                <item name="/config/welcome_message" xsi:type="string">store_id</item>
            </argument>
        </arguments>
    </virtualType>
    <!--Config Model itself-->
    <type name="Unit1\CustomConfig\Model\Config">
        <arguments>
            <argument name="reader"
xsi:type="object">Unit1\CustomConfig\Model\Config\Reader</argument>
```

```xml
            <argument name="cacheId" xsi:type="string">welcome_messages</argument>
        </arguments>
    </type>

</config>
```

## Phase 3

### 3.1) Create storefront page route etc/frontend/routes.xml:

```xml
<?xml version="1.0"?>
<!--
/**
 *
 * Copyright © 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="custom_config" frontName="custom_config">
            <module name="Unit1_CustomConfig"/>
        </route>
    </router>
</config>
```

### 3.2) Create an action class:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit1\CustomConfig\Controller\Test;

use Magento\Framework\App\Action\Context;
use Magento\Framework\App\Action\Action;
use Magento\Framework\Controller\ResultFactory;

/**
 * Class Index
 * @package Unit1\CustomConfig\Controller\Test\Index
 */
class Index extends Action
{

    /**
```

```
     * @var Unit1\CustomConfig\Model\Config
     */
    private $customConfig;

    /**
     * Index constructor.
     * @param Context $context
     */
    public function __construct(
        Context $context,
        \Unit1\CustomConfig\Model\Config $customConfig
    )
    {
        $this->customConfig = $customConfig;
        return parent::__construct($context);
    }

    /**
     * @return \Magento\Framework\View\Result\Page
     */
    public function execute()
    {
        $storeId = 2;
        $storeWelcomeMsg = $this->customConfig->get('messages/' . $storeId . '/message');

        $result = $this->resultFactory->create(ResultFactory::TYPE_RAW);
        $result->setContents($storeWelcomeMsg);

        return $result;
    }
}
```

Now, when clean your cache and enable module – you see content of your config file on the page
http://domain.name/custom_config/test/index.

# Unit 2. Request Flow

## Module 2. Request Flow Overview

**2.2.1. (module name: Unit2_FlushOutput) 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`:

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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_front_send_response_before">
        <observer name="unit2_flushoutput"
instance="Unit2\FlushOutput\Observer\LogPageOutput" shared="false" />
    </event>
</config>
```

2. Create an observer class:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\FlushOutput\Observer;

use Magento\Framework\Event\ObserverInterface;

class LogPageOutput implements ObserverInterface
{
    /**
     * @var \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();
        $body = substr($body, 0, 1000);
        $this->_logger->info("--------\n\n\n BODY \n\n\n ". $body, []);
    }
}
```

Note! If you are using `$this->_logger`->addDebug method, first you should turn log to file option on located at stores->configuration->developer->debug.

# Module 3. Request Routing

## 2.3.1. (module name: Unit2_RoutersLog) Create an extension that logs into the file list of all available routers into a file.

**Solution**

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

```
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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:ObjectManager/etc/config.xsd">
    <preference for="Magento\Framework\App\FrontController"
type="Unit2\RoutersLog\Test\App\FrontController" />
</config>
```

2. Implement a front controller class:

```
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\RoutersLog\Test\App;
```

```php
use Magento\Framework\App\RequestInterface;
use Magento\Framework\App\Request\Http as HttpRequest;
use Magento\Framework\App\ActionInterface;
use Magento\Framework\App\Request\ValidatorInterface as RequestValidator;
use Magento\Framework\App\ObjectManager;
use Magento\Framework\App\RouterListInterface;
use Magento\Framework\App\ResponseInterface;
use Magento\Framework\Message\MessageInterface as MessageManager;
use Psr\Log\LoggerInterface;

/**
 * Class FrontController
 * @package Unit2\RoutersLog\Test\App
 */
class FrontController extends \Magento\Framework\App\FrontController
{
    /**
     * @var RequestValidator
     */
    private $requestValidator;

    /**
     * @var LoggerInterface
     */
    private $logger;

    public function __construct(
        RouterListInterface $routerList,
        ResponseInterface $response,
        ?RequestValidator $requestValidator = null,
        ?MessageManager $messageManager = null,
        ?LoggerInterface $logger = null
    ){
        $this->logger = $logger
            ?? ObjectManager::getInstance()->get(LoggerInterface::class);
        parent::__construct($routerList, $response, $requestValidator, $messageManager,
$logger);
    }

    /**
     * Perform action and generate response
     *
     * @param Magento\Framework\App\RequestInterface|HttpRequest $request
     * @return ResponseInterface|ResultInterface
     * @throws \LogicException
     */
    public function dispatch(RequestInterface $request)
    {
```

```
        $routerList = [];
        foreach ($this->_routerList as $router) {
            $routerList[] = $router;
        }
        $routerList = array_map(function ($item) {
            return get_class($item);
        }, $routerList);
        $routerList = "\n\r" . implode("\n\r", $routerList);
        $this->logger->info("Magento2 Routers List:" . $routerList);

        return parent::dispatch($request);
    }
}
```

## 2.3.2. (module name: Unit2_RouterDash) Create a new router that "understands" URLs like `/frontNameactionPath-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:

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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:ObjectManager/etc/config.xsd">
    <type name="Magento\Framework\App\RouterList">
        <arguments>
            <argument name="routerList" xsi:type="array">
                <item name="routerdash" xsi:type="array">
                    <item name="class"
xsi:type="string">Unit2\RouterDash\Controller\Router</item>
                    <item name="disable" xsi:type="boolean">false</item>
                    <item name="sortOrder" xsi:type="string">70</item>
                </item>
            </argument>
        </arguments>
    </type>
</config>
```

2. Create a router class:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
```

```
 * See COPYING.txt for license details.
 */
namespace Unit2\RouterDash\Controller;

/**
 * Class Router
 * @package Unit2\RouterDash\Controller
 */
class Router implements \Magento\Framework\App\RouterInterface
{
    /**
     * @var \Magento\Framework\App\ActionFactory
     */
    protected $actionPath;

    /**
     * Router constructor.
     * @param \Magento\Framework\App\ActionFactory $actionFactory
     */
    public function __construct(\Magento\Framework\App\ActionFactory $actionFactory) {
        $this->actionPath = $actionFactory;
    }

    /**
     * @param \Magento\Framework\App\RequestInterface $request
     * @return \Magento\Framework\App\ActionInterface|null
     */
    public function match(\Magento\Framework\App\RequestInterface $request) {
        $testCategory = 'id/6';
        $info = $request->getPathInfo();
        if (preg_match("%^/(.*?)-(.*?)-(.*?)$%", $info, $m)) {
            $request->setPathInfo(sprintf("/%s/%s/%s/%s", $m[1], $m[2], $m[3],
$testCategory));
            return $this->actionPath->create('Magento\Framework\App\Action\Forward',
                ['request' => $request]);
        }
        return null;
    }
    }
```

3.  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. (module name: Unit2_CustomNotFound) 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 `di.xml`:

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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:ObjectManager/etc/config.xsd"
>
    <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">Unit2\CustomNotFound\Controller\NoRouteHandler</item>
                    <item name="sortOrder" xsi:type="string">9</item>
                </item>
            </argument>
        </arguments>
    </type>
</config>
```

2.  Create a handler class:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\CustomNotFound\Controller;

class NoRouteHandler implements
\Magento\Framework\App\Router\NoRouteHandlerInterface
{
    public function process(\Magento\Framework\App\RequestInterface $request)
    {
    if ($request->getFrontName() == "admin") {
            return false;
        }
```

```
            $moduleName = 'cms';
            $controllerName = 'index';
            $actionName = 'index';
            $request
                ->setModuleName($moduleName)
                ->setControllerName($controllerName)
                ->setActionName($actionName);

            return true;
        }
    }
```

# Module 5. Working with Controllers

## 2.5.1. (module name: Unit2_HelloWorldController) Create a frontend controller that renders "HELLO WORLD"

**Solution**

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

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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="helloworldcontroller" frontName="helloworldcontroller">
            <module name="Unit2_HelloWorldController" />
        </route>
    </router>
</config>
```

2.  Create an action class:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\HelloWorldController\Controller\Action;

/**
 * Class Index
 * @package Unit2\HelloWorldController\Controller\Action
 */
class Index extends \Magento\Framework\App\Action\Action
{
```

```
    /**
     * @var \Magento\Framework\View\Result\PageFactory
     */
    protected $_pageFactory;

    /**
     * Index constructor.
     * @param \Magento\Framework\App\Action\Context $context
     * @param \Magento\Framework\View\Result\PageFactory $pageFactory
     */
    public function __construct(
        \Magento\Framework\App\Action\Context $context,
        \Magento\Framework\View\Result\PageFactory $pageFactory
    )
    {
        $this->_pageFactory = $pageFactory;
        return parent::__construct($context);
    }

    /**
     * @return \Magento\Framework\Controller\ResultInterface
     */
    public function execute()
    {
        $result = $this->resultFactory-
>create(\Magento\Framework\Controller\ResultFactory::TYPE_RAW);
        $result->setContents('Hello World');
        return $result;
    }
}
```

You can view your new page at the URL: `helloworldcontroller/action/index`

## 2.5.2. (module name: Unit2_CatalogProductPlugin and Unit2_CatalogProductPreference) 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`:

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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:ObjectManager/etc/config.xsd"
>

    <type name="Magento\Catalog\Controller\Product">
        <plugin name="product-view-controller-plugin"
type="Unit2\CatalogProductPlugin\Controller\Product\View" sortOrder="5"
```

```
            disabled="false" />
        </type>

    </config>
```

***Or***

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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:ObjectManager/etc/config.xsd"
>

        <preference for="Magento\Catalog\Controller\Product\View"
    type="Unit2\CatalogProductPreference\Controller\Product\View" />

    </config>
```

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

2. Now you can implement your preference/plugin:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\CatalogProductPlugin\Controller\Product;

/**
 * Class View
 * @package Unit2\CatalogProductPlugin\Controller\Product
 */
class View extends \Magento\Framework\App\Action\Action
{

    /**
     * @return
\Magento\Framework\App\ResponseInterface|\Magento\Framework\Controller\ResultInterface
     */
    public function execute()
    {
        return $this->resultFactory->create('raw')->setContents(' echo plugin ');
```

```
    }

    /**
     * @param \Magento\Catalog\Controller\Product\View $controller
     * @param $result
     * @return mixed
     */
    public function afterExecute(\Magento\Catalog\Controller\Product\View $controller,
$result)
    {
        /**
         * Custom code goes here
         */

        return $result;
    }
}
```

## *Or*

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\CatalogProductPreference\Controller\Product;

use \Magento\Framework\Controller\ResultFactory;

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

    public function execute()
    {
        $rawResult = $this->resultFactory->create(ResultFactory::TYPE_RAW);
        $rawResult->setContents('Hello world');
        return $rawResult;
    }

}
```

### 2.5.3. (module name: Unit2_Secret) 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
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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="unit2secret" frontName="unit2secret">
            <module name="Unit2_Secret" before="Magento_Backend" />
        </route>
    </router>
</config>
```

2.  Create an action class:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\Secret\Controller\Adminhtml\Action;

/**
 * Class Index
 * @package Unit2\Secret\Controller\Adminhtml\Action
 */
class Index extends \Magento\Backend\App\Action
{
    /**
     * execute method
     */
    public function execute()
    {
        $result = $this->resultFactory-
>create(\Magento\Framework\Controller\ResultFactory::TYPE_RAW);
        $result->setContents('Hello World!');
        return $result;
    }
```

```
    /**
     * @return int
     */
    protected function _isAllowed() {
        $secret = $this->getRequest()->getParam('secret');
        return isset($secret) && (int)$secret==1;
    }

    /**
     * @return true
     */
    protected function _processUrlKeys() {
        return true;
    }
}
```

## 2.5.4. (module_name: Unit2_HelloWorldRedirect) 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).

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

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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="unit2helloworldredirect" frontName="unit2helloworldredirect">
            <module name="Unit2_HelloWorldRedirect" before="Magento_Backend" />
        </route>
    </router>
</config>
```

2.  Create an action class:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\HelloWorldRedirect\Controller\Adminhtml\Action;

/**
```

```
 * Class HelloWorld
 * @package Unit2\HelloWorldRedirect\Controller\Adminhtml\Action
 */
class Helloworld extends \Magento\Backend\App\Action
{
    /**
     * execute method
     */
    public function execute()
    {
        $this->_redirect('catalog/category/edit/id/38');
    }

    /**
     * Link must be generated by server side
     * It's only for education purpose!
     *
     * @return bool
     */
    public function _processUrlKeys()
    {
        return true;
    }
}
```

Of course, you can put any other `categoryId` that exists in your system.


# 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=' helloworldcontroller.html', target_path=' helloworldcontroller/action/index',redirect_type=0, store_id=1, is_autogenerated=0;

# Unit 3. Rendering

## Module 5. Block Architecture & Life Cycle

**3.5.1. (module name: Unit3_HelloWorldBlock) Create a block extending `AbstractBlock` and implement the `_toHtml()` method. Render that block in the new controller.**

**Solution**

1. Create the block:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit3\HelloWorldBlock\Block;

/**
 * Class Test
 * @package Unit3\HelloWorldBlock\Block
 */
class Test extends \Magento\Framework\View\Element\AbstractBlock
{
    /**
     * @return string
     */
    protected function _toHtml()
    {
        return "<b>Hello world from the block!</b>";
    }
}
```

2. Create an action class:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit3\HelloWorldBlock\Controller\Block;

/**
 * Class Index
 * @package Unit3\HelloWorldBlock\Controller\Block
 */
class Index extends \Magento\Framework\App\Action\Action
{
    protected $_pageFactory;
```

```php
    public function __construct(
        \Magento\Framework\App\Action\Context $context,
        \Magento\Framework\View\Result\PageFactory $pageFactory
    ) {
        $this->_pageFactory = $pageFactory;
        parent::__construct($context);
    }

    /**
     * return \Magento\Framework\App\ResponseInterface |
\Magento\Framework\Controller\ResultInterface
     */
    public function execute()
    {
        $layout = $this->_pageFactory->create()->getLayout();
        $block = $layout->createBlock('Unit3\HelloWorldBlock\Block\Test');
        $result = $this->resultFactory-
>create(\Magento\Framework\Controller\ResultFactory::TYPE_RAW);
        $result->setContents($block->toHtml());
        return $result;
    }
}
```

## 3.5.2. (module name: Unit3_TestBlock) Create and render a text block in the controller.

**Solution**

1.  Create an action class:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit3\TestBlock\Controller\Block;

use Magento\Framework\View\Element\Template;

/**
 * Class Text
 * @package Unit3\TestBlock\Controller\Block
 */
class Text extends \Magento\Framework\App\Action\Action
{
    /**
     * @var \Magento\Framework\View\Result\PageFactory
     */
    protected $_pageFactory;
```

```php
    /**
     * Text constructor.
     * @param \Magento\Framework\App\Action\Context $context
     * @param \Magento\Framework\View\Result\PageFactory $pageFactory
     */
    public function __construct(
        \Magento\Framework\App\Action\Context $context,
        \Magento\Framework\View\Result\PageFactory $pageFactory
    ) {
        $this->_pageFactory = $pageFactory;
        parent::__construct($context);
    }


    /**
     * @return \Magento\Framework\Controller\ResultInterface
     */
    public function execute()
    {
        $block = $this->_pageFactory->create()->getLayout()-
>createBlock('Magento\Framework\View\Element\Text');
        $block->setText("Hello World From a New Module!");
        $result = $this->resultFactory-
>create(\Magento\Framework\Controller\ResultFactory::TYPE_RAW);
        $result->setContents($block->toHtml());

        return $result;
    }
}
```

2. Create action route.

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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="unit3testblock" frontName="unit3testblock">
            <module name="Unit3_TestBlock" />
        </route>
    </router>
</config>
```

### 3.5.3. (module name: Unit3_ProductViewDescriptionPlugin) 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:

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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:ObjectManager/etc/config.xsd">

    <type name="Magento\Catalog\Block\Product\View\Description">
        <plugin name="product-view-description-plugin"

type="Unit3\ProductViewDescriptionPlugin\Block\Product\View\Description" />
    </type>

</config>
```

2. Create a plugin class:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit3\ProductViewDescriptionPlugin\Block\Product\View;

/**
 * Class Description
 * @package Unit3\ProductViewDescriptionPlugin\Block\Product\View
 */
class Description extends \Magento\Framework\View\Element\Template
{
    /**
     * @param \Magento\Catalog\Block\Product\View\Description $description
     */
    public function beforeToHtml(\Magento\Catalog\Block\Product\View\Description $description)
    {
        $description->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. (module name: Unit3_TemplateBlock) Create a template block and a custom template file for it. Render the block in the controller.

**Solution**

1. Create the block:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit3\TemplateBlock\Block;

/**
 * Class Template
 * @package Unit3\TemplateBlock\Block
 */
class Template extends \Magento\Framework\View\Element\Template
{

}
```

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

2. Create a template file Unit3/TemplateBlock/view/frontend/templates/template.phtml:
`<?= __("Hello from template"); ?>`.

3. Create an action class:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit3\TemplateBlock\Controller\Action;

/**
 * Class Index
 * @package Unit3\TemplateBlock\Controller\Action
 */
class Index extends \Magento\Framework\App\Action\Action
{
```

```php
        /**
         * @var \Magento\Framework\View\Result\PageFactory
         */
        protected $_pageFactory;

        /**
         * Index constructor.
         * @param \Magento\Framework\App\Action\Context $context
         * @param \Magento\Framework\View\Result\PageFactory $pageFactory
         */
        public function __construct(
            \Magento\Framework\App\Action\Context $context,
            \Magento\Framework\View\Result\PageFactory $pageFactory
        ) {
            $this->_pageFactory = $pageFactory;
            parent::__construct($context);
        }

        /**
         * @return \Magento\Framework\Controller\ResultInterface
         */
        public function execute()
        {
            $block = $this->_pageFactory->create()->getLayout()-
>createBlock('Unit3\TemplateBlock\Block\Template');
            $block->setTemplate('template.phtml');

            $result = $this->resultFactory-
>create(\Magento\Framework\Controller\ResultFactory::TYPE_RAW);
            $result->setContents($block->toHtml());

            return $result;
        }
    }
```

## 3.6.3. (Unit3_ProductViewDescriptionBlock) Customize the Catalog\Block\Product\View\Description block and assign a custom template to it.

**Solution**

1. Using the same declaration as in 3.5.3, change the `beforeToHtml` method to:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit3\ProductViewDescriptionBlock\Block\Product\View;

/**
 * Class Description
 * @package Unit3\ProductViewDescriptionBlock\Block\Product\View
 */
```

```
class Description extends \Magento\Framework\View\Element\Template
{
    /**
     * @param \Magento\Catalog\Block\Product\View\Description $description
     */
    public function beforeToHtml(\Magento\Catalog\Block\Product\View\Description
$description)
    {
        $description->setTemplate('Unit3_ProductViewDescriptionBlock::description.phtml');
    }
}
```

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

```
<h1>Custom description template!</h1>
<div>
    Product description:
    <?= $block->getProduct()->getDescription();?>
</div>
```

3. Create etc/frontend/di.xml file

```
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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:ObjectManager/etc/config.xsd">

    <type name="Magento\Catalog\Block\Product\View\Description">
        <plugin name="product-view-description-block"
                type="Unit3\ProductViewDescriptionBlock\Block\Product\View\Description"
sortOrder="11" />
    </type>

</config>
```

# 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. (module name: Unit3_Layout) Add a `default.xml` layout file to your 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.

**Solution**

1. Create view/frontend/layout/default.xml

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configuration.xsd"
>
    <body>
        <referenceContainer name="content.top">
            <block class="Magento\Framework\View\Element\Template" name="custom_layout"
template="Unit3_Layout::template.phtml" />
        </referenceContainer>
    </body>
</page>
```

2. Create template file template.phtml.
```php
<div style="background: <?php echo $this->getData('background_color'); ?>;
             color: <?php echo $this->getData('color');  ?>">
    <?= __('some text for all pages') ?>
</div>
```

Please note – exercise solution contains layout directive to remove "custom_layout" node in order to avoid page brake. Remove that line to see the template.

## 3.8.2. (module name: Unit3_Layout) Create a new controller action (ex: unit3layout/layout/onepage).

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

**Solution**

1. Create your action class:

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit3\Layout\Controller\Layout;

/**
 * Class Onepage
 * @package Unit3\Layout\Controller\Layout
 */
```

```
class Onepage extends \Magento\Framework\App\Action\Action
{
    /**
     * @var \Magento\Framework\View\Result\PageFactory
     */
    protected $_pageFactory;

    /**
     * Onepage constructor.
     * @param \Magento\Framework\App\Action\Context $context
     * @param \Magento\Framework\View\Result\PageFactory $pageFactory
     */
    public function __construct(
        \Magento\Framework\App\Action\Context $context,
        \Magento\Framework\View\Result\PageFactory $pageFactory)
    {
        $this->_pageFactory = $pageFactory;
        return parent::__construct($context);
    }

    /**
     * @return \Magento\Framework\View\Result\Page
     */
    public function execute()
    {
        return $this->_pageFactory->create();
    }
}
```

2. Create your layout xml file

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" layout="1column"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configuration.xsd"
>
    <body>
        <referenceBlock name="page.main.title">
            <action method="setPageTitle">
                <argument translate="true" name="title" xsi:type="string">Training Advanced
Text</argument>
            </action>
        </referenceBlock>
    </body>
</page>
```

### 3.8.3. (module name: Unit3_Layout) 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
4. Confirm that the background color is displayed.

**Solution**

1. Add next node to your layout xml file

```
<referenceContainer name="content">
            <block class="Magento\Framework\View\Element\Template" name="one_page_layout"
template="Unit3_Layout::onepage.phtml">
                <arguments>
                    <argument name="background_color" xsi:type="string">#000</argument>
                    <argument name="color" xsi:type="string">lightskyblue</argument>
                </arguments>
            </block>
</referenceContainer>
```

2. Create onepage.phtml template and put below:

```
<div
    style="background: <?php echo $this->getData('background_color'); ?>;
    color: <?php $this->getData('color');  ?>">
        <?php echo __('one page text'); ?>
</div>
```

### 3.8.4. (module name: Unit3_Layout) Change the block color to orange on the product detail pages only.

**Solution**
1. Create catalog_product_view.xml file

```
<?xml version="1.0"?>
<!--
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configuration.xsd"
>
    <body>
        <referenceBlock name="custom_layout">
            <arguments>
                <argument name="background_color" xsi:type="string">#000</argument>
                <argument name="color" xsi:type="string">orange</argument>
            </arguments>
        </referenceBlock>
    </body>
</page>
```

2. Add css styles to your template.phtml file:

```
<div
        style="background: <?php echo $this->getData('background_color'); ?>;
               color: <?php echo $this->getData('color');  ?>">
<?php echo __('some text for all pages') ?>
</div>
```

## 3.8.5. (module name: Unit3_Layout) On category pages, move the exercise block to the bottom of the left column.

**Solution**
Create catalog_category_view.xml file:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configurati
on.xsd">
    <body>
        <referenceContainer name="sidebar.main">
            <block class="Magento\Framework\View\Element\Template"
name="custom_category_layout" template="Unit3_Layout::template.phtml" />
        </referenceContainer>
    </body>
</page>
```

## 3.8.6. (module name: Unit3_Layout) On the custom action you just added, remove the custom block from the content.top container. (See Exercise 3.8.1.)

**Solution**
Add next node to your default.xml file instead previous:

```
<referenceBlock name="custom_layout" remove="true" />
```

### 3.8.7. (module name: Unit3_Layout) 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.

**Solution**

Add next node to your layout xml file:

```
<referenceBlock name="header.links">
    <block class="Magento\Framework\View\Element\Html\Link" name="test-link">
        <arguments>
            <argument name="label" xsi:type="string" translate="true">Test
Link1</argument>
            <argument name="path" xsi:type="string"
translate="true">testlink1</argument>
        </arguments>
    </block>
</referenceBlock>
```

# Unit 4. Databases & Entity-Attribute-Value (EAV)

## Module 1. Declarative Schema

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

### 4.1.1 (solution module Unit4_VendorEntity) Create a table `vendor_entity` using declarative schema approach.

- Create a new table, `vendor_entity`, with the following fields:
    - `vendor_id`
    - `code`
    - `contact`

**Solution**

1. Create etc/db_schema.xml for defining vendor's entity table.

```
<schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="urn:magento:framework:Setup/Declaration/Schema/etc/schema.xsd">
    <table name="vendor_entity" resource="default" engine="innodb" comment="Vendor entity table">
        <column xsi:type="int" name="vendor_id" padding="10" unsigned="true" nullable="false" identity="true" comment="Vendor Primary field"/>
        <column xsi:type="text" name="code" nullable="false"  comment="Vendor Name"/>
        <column xsi:type="text" name="contact" nullable="false"  comment="Vendor Contact"/>
        <constraint xsi:type="primary" referenceId="PRIMARY">
            <column name="vendor_id"/>
        </constraint>
    </table>
</schema>
```

2. Generate db_schema_whitelist.json by running "php bin/magento setup:db-declaration:generate-whitelist"

### 4.1.2 (solution module Unit4_VendorEntity) Modify `vendor_entity` table declarative schema approach.

- Create the `VendorColumn` class.
- Add an additional column `goods_type` to the `vendor_entity` table using DDL adapter methods.
- Run the appropriate console command.
- Verify that it works.

**Solution**

```php
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit4\VendorEntity\Setup\Patch\Schema;


use Magento\Framework\Setup\Patch\PatchInterface;
use Magento\Framework\Setup\Patch\SchemaPatchInterface;
use Magento\Framework\Setup\SchemaSetupInterface;


/**
 * @package Unit4\VendorEntity\Setup
 */
class VendorColumn implements SchemaPatchInterface
{
    /**
     * @var \Magento\Framework\Setup\SchemaSetupInterface;
     */
    protected $moduleSchemaSetup;


    /**
     * VendorColumn constructor.
     * @param SchemaSetupInterface $moduleSchemaSetup
     */
    public function __construct(SchemaSetupInterface $moduleSchemaSetup)
    {
        $this->moduleSchemaSetup = $moduleSchemaSetup;
    }


    /**
     * @return SchemaPatchInterface|void
     */
    public function apply()
    {
        $this->moduleSchemaSetup->startSetup();


        $this->moduleSchemaSetup->getConnection()->addColumn('vendor_entity',
'goods_type',
            [
                'type' => \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,
                'length' => 64,
                'unsigned' => true,
                'nullable' => false,
                'comment' => 'Vendor goods type'
            ]
```

```
            );

            $this->moduleSchemaSetup->endSetup();
        }


    /**
     * @return array|string[]
     */
    public static function getDependencies()
    {
        return [];
    }


    /**
     * @return array|string[]
     */
    public function getAliases()
    {
        return [];
    }
}
```

## 4.1.3. (solution module Unit4_VendorEntity) Create a data patch script to set a few sample records to the `vendor_entity` table.

- Create the `Vendors` class.
- Define a fixture vendor to be installed along with your module.
- Execute the appropriate console command.
- Verify that it works.

**Solution**

```php
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit4\VendorEntity\Setup\Patch\Data;

use Magento\Framework\Setup\ModuleDataSetupInterface;
use Magento\Framework\Setup\Patch\DataPatchInterface;
use Magento\Framework\Setup\Patch\PatchInterface;

/**
 * @package Unit4\VendorEntity\Setup
 */
```

```php
class Vendors implements DataPatchInterface
{
    /**
     * @var ModuleDataSetupInterface
     */
    protected $moduleDataSetup;


    /**
     * Vendors constructor.
     * @param ModuleDataSetupInterface $moduleDataSetup
     */
    public function __construct(ModuleDataSetupInterface $moduleDataSetup)
    {
        $this->moduleDataSetup = $moduleDataSetup;
    }


    /**
     * @return DataPatchInterface|void
     */
    public function apply()
    {
        $this->moduleDataSetup->startSetup();


        $this->moduleDataSetup->getConnection()->insert('vendor_entity',
            [
                'code'    => 'Auchan',
                'contact' => '38011122333',
                'goods_type'    => 'food'
            ]
        );


        $this->moduleDataSetup->endSetup();
    }


    /**
     * @return array|string[]
     */
    public static function getDependencies()
    {
        return [];
    }


    /**
     * @return array|string[]
     */
    public function getAliases()
    {
```

```
        return [];
    }
}
```

3.  Run "bin/magento set:up".

**Note!** Patches is applied only once, you can check if your patch has been applied at table patch_list. When db_schema.xml applied each time if the table for example were delete.

# Module 2. Databases Overview

## List Root Categories by Store

## 4.2.1. (solution module Unit4_RootCategories) Echo the list of all store views and associated root categories.

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

  `Magento\Store\Model\Store::getRootCategoryId()`

- Create a category API data object and load category by ID.
- Add the category name attribute to the result array.
- Display stores with the associated root category names.

**Solution**

1.  Create new page to display the list.
```php
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */

namespace Unit5\RootCategories\Controller\Index;

/**
 * Class Index
 * @package Unit4\RootCategories\Controller\Index
 */
class Index extends \Magento\Framework\App\Action\Action
{
    /**
     * @return \Magento\Framework\View\Result\Page
     */
    public function execute()
    {
        return $this->resultFactory->create(ResultFactory::TYPE_PAGE);
    }
}
```

2. routes.xml

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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="rootcategories" frontName="rootcategories">
            <module name="Unit5_RootCategories"/>
        </route>
    </router>
</config>
```

3. Page layout xml file.

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configuration.xsd">
    <body>
        <referenceContainer name="content">
            <block class="Magento\Framework\View\Element\Template"
name="stores.listing" template="Unit4_RootCategories::categories.phtml" >
        <arguments>
                <argument name="view_model"
        xsi:type="object">Unit4\RootCategories\ViewModel\StoreList</argument>
        </arguments>
    </block>
        </referenceContainer>
    </body>
</page>
```

4. ViewModel file.

```php
<?php

namespace Unit4\RootCategories\ViewModel;

use \Magento\Framework\View\Element\Block\ArgumentInterface;
use \Magento\Catalog\Model\CategoryFactory;
use \Magento\Store\Model\StoreManager;

/**
 * Class StoresList
 * @package Unit4\RootCategories\ViewModel
 */
class StoresList implements ArgumentInterface
{
    /**
```

```php
     * @var StoreManager
     */
    protected $storeManager;

    /**
     * @var CategoryFactory
     */
    protected $categoryFactory;

    /**
     * StoresList constructor.
     * @param CategoryFactory $categoryFactory
     * @param StoreManager $storeManager
     */
    public function __construct(
        CategoryFactory $categoryFactory,
        StoreManager $storeManager
    ) {
        $this->categoryFactory = $categoryFactory;
        $this->storeManager    = $storeManager;
    }

    /**
     * @return string
     */
    public function getRootCategories()
    {
        $storesList = $this->storeManager->getStores();
        $catalogCategory = $this->categoryFactory->create();

        $stores = [];
        foreach ($storesList as $store) {
            $categoryName = $catalogCategory->load($store->getRootCategoryId())-
>getName();
            $stores[] = [
                'store_name' => $store->getName(),
                'root_category_name' => $categoryName
            ];
        }

        $stores = array_map(function($item)
        {
            $string = '<b>STORE</b> ' . $item['store_name'] . '<br>';
            $string .= ' <b>ROOT CATEGORY</b> ' . $item['root_category_name'] . '<br>
<br>';

            return $string;
        }, $stores);

        $response = implode('', $stores);
        return $response;
    }

}
```

5.  templates/categories.phtml file.

```php
<?php $viewModel = $this->getViewModel(); ?>
<?php echo $viewModel->getRootCategories(); ?>
```

# 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. (solution module Unit4_ProductSave) Log every product save operation and specify the product ID and the data that has been changed.

- Create an empty observer class that implements `ObserverInterface`.
- Create a logger object from \Psr\Log\LoggerInterface in the observer class.
- Implement the execute function.
- Create an `events.xml` file and tie the observer to the "catalog_product_save_after" event.

**Solution**

1.  Create observer class.

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

namespace Unit4\ProductSave\Observer;
use Magento\Framework\Event\ObserverInterface;

/**
 * Class LogProductSave
 * @package Unit4\ProductSave\Observer
 */
class LogProductSave implements ObserverInterface
{
    /**
     * @var null|\Psr\Log\LoggerInterface
     */
    protected $_logger = null;

    /**
     * LogProductSave constructor.
     */
    public function __construct(\Psr\Log\LoggerInterface $logger)
    {
        $this->_logger = $logger;
    }

    /**
     * @param \Magento\Framework\Event\Observer $observer
     */
    public function execute(\Magento\Framework\Event\Observer $observer)
    {
        $product = $observer->getEvent()->getProduct();
```

```
        if ($product->getId() && ($product->isDataChanged() || $product->isObjectNew())) {
            $logMessage  = PHP_EOL . 'Product Saving Log...' . PHP_EOL;

            foreach ($product->getData() as $key => $dataItem) {
                if ((is_string($dataItem) || is_int($dataItem)) && $dataItem != $product-
>getOrigData($key)) {
                    $logMessage .= $key . ' = ' . $dataItem . PHP_EOL;
                }
            }
            $this->_logger->info($logMessage);
        }
    }
}
```

2. Assign observer to "catalog_product_save_after" event.

```
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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="catalog_product_save_after" >
        <observer name="LogProductSave"
instance="Unit4\ProductSave\Observer\LogProductSave"/>
    </event>
</config>
```

# Module 5. Attribute Management

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

- Add it to an attribute set.
- Check that it appears on the product edit page.
- 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 Admin.
2. Select **Stores > Attributes > Product** in the main navigation.
3. Click the **Add New Attribute** button.
4. Enter an attribute label, for example: **Flavors.**
5. Select the tab **Frontend Properties**.
6. Set **Visible on Catalog Pages on Frontend** to **Yes**.
7. Click **Save Attribute**.
8. Select **Stores > Attributes > Attribute Sets** in the main navigation.
9. In the list, select an Attribute Set, for example the **Bag** attribute set.

10. Drag and drop the **Flavors** attribute icon from the right **Unassigned Attributes** column onto the **Product Details** attribute group folder icon.
11. Confirm that the **Flavors** attribute icon now is listed within the **Product Details** attribute group.
12. Click **Save** button.
13. Select **Catalog > Inventory > Products** in the main navigation.
14. In the **Attribute Set** column filter dropdown, select **Bag** 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 **Flavors** attribute field is displayed on the **Product Details** form.
17. Enter a value for the **Flavors** attribute, for example **Strawberry.**
18. Click the **Save** button.
19. Open the product in the Magento storefront.
20. Select the **More Information** tab.
21. Confirm that the new attribute and the value you gave it are displayed.

## 4.5.2. (solution module Unit4_TextInput) Create a text input attribute.

- Create a module. Create a data patch class.
- For adding an attribute, create an instance of the `CategorySetup` class by calling its factory create method and passing the setup object as a param. Call `addAttribute` on it.
- Enable the module and apply the setup changes.
- Open a product in the Admin interface and confirm that the new attribute exists and can be set on a store view level.
- Visit a product on the storefront and confirm that the new attribute is visible there, too.

**Solution**

1. Create setup data patch as it is kind of data manipulation operation.

```php
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit4\TextInput\Setup\Patch\Data;

use Magento\Framework\Setup\ModuleDataSetupInterface;
use Magento\Framework\Setup\Patch\DataPatchInterface;
use Magento\Framework\Setup\Patch\PatchInterface;

use Magento\Catalog\Setup\CategorySetup;
use Magento\Catalog\Setup\CategorySetupFactory;
use Magento\Catalog\Model\Product;
use Magento\Catalog\Model\ResourceModel\Eav\Attribute as CatalogAttribute;

/**
 * @package Unit4\TextInput\Setup
 */
class CategoryAttr implements DataPatchInterface
{
    /**
     * @var CategorySetupFactory
```

```php
     */
    protected $categorySetupFactory;


    /**
     * @var ModuleDataSetupInterface
     */
    protected $moduleDataSetup;


    /**
     * CategoryAttr constructor.
     * @param CategorySetupFactory $categorySetupFactory
     * @param ModuleDataSetupInterface $moduleDataSetup
     */
    public function __construct(
        CategorySetupFactory $categorySetupFactory,
        ModuleDataSetupInterface $moduleDataSetup
    ) {
        $this->categorySetupFactory = $categorySetupFactory;
        $this->moduleDataSetup = $moduleDataSetup;
    }


    /**
     * @return DataPatchInterface|void
     */
    public function apply()
    {
        $this->moduleDataSetup->startSetup();
        /** @var CategorySetup $catalogSetup */
        $catalogSetup = $this->categorySetupFactory->create(['setup' => $this->moduleDataSetup]);
        $catalogSetup->addAttribute(Product::ENTITY, 'flavor', [
            'label'            => 'Flavor',
            'visible_on_front'  => 1,
            'required'         => 0,
            'global'           => CatalogAttribute::SCOPE_STORE
        ]);

        $this->moduleDataSetup->endSetup();
    }


    /**
     * @return array|string[]
     */
    public static function getDependencies()
    {
        return [];
    }


    /**
     * @return array|string[]
     */
    public function getAliases()
    {
        return [];
    }
}
```

2. Apply patch by running "php bin/magento set:up"

### 4.5.3. (solution module Unit4_MultiSelect) Create a multiselect product attribute from an upgrade data method.

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

**Solution**

1. Write category new attribute data patch.

```php
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit4\MultiSelect\Setup\Patch\Data;

use Magento\Framework\Setup\ModuleDataSetupInterface;
use Magento\Framework\Setup\Patch\DataPatchInterface;
use Magento\Framework\Setup\Patch\PatchInterface;

use Magento\Catalog\Model\Product;
use Magento\Catalog\Setup\CategorySetupFactory;
use Magento\Catalog\Model\ResourceModel\Eav\Attribute as CatalogAttribute;


/**
 * Class CategoryAttr
 * @package Unit4\MultiSelect\Setup\Patch\Data
 */
class CategoryAttr implements DataPatchInterface
{
    /**
     * @var CategorySetupFactory
     */
    protected $categorySetupFactory;

    /**
     * @var ModuleDataSetupInterface
     */
    protected $moduleDataSetup;

    /**
     * CategoryAttr constructor.
     * @param CategorySetupFactory $categorySetupFactory
     * @param ModuleDataSetupInterface $moduleDataSetup
     */
    public function __construct(
        CategorySetupFactory $categorySetupFactory,
        ModuleDataSetupInterface $moduleDataSetup
    ) {
```

```
        $this->categorySetupFactory = $categorySetupFactory;
        $this->moduleDataSetup = $moduleDataSetup;
    }

    /**
     * @return DataPatchInterface|void
     */
    public function apply()
    {
        $this->moduleDataSetup->startSetup();

        $catalogSetup = $this->categorySetupFactory->create(['setup' => $this-
>moduleDataSetup]);
        $attrParams = [
            'type' => 'text',
            'label' => 'Custom multiselect attribute',
            'input' => 'multiselect',
            'required' => 0,
            'visible_on_front' => 1,
            'global' => CatalogAttribute::SCOPE_STORE,
            'backend' => 'Magento\Eav\Model\Entity\Attribute\Backend\ArrayBackend',
            'option' => ['values' => [
                'left',
                'right',
                'up',
                'down'
            ]]
        ];
        $catalogSetup->addAttribute(Product::ENTITY, 'custom_multiselect', $attrParams);

        $this->moduleDataSetup->endSetup();
    }

    /**
     * @return array|string[]
     */
    public static function getDependencies()
    {
        return [];
    }

    /**
     * @return array|string[]
     */
    public function getAliases()
    {
        return [];
    }
}
```

2.  Apply patch by running "php bin/magento set:up". Verify the result.

## 4.5.4. (solution module Unit4_MultiSelect) Customize the rendering of the values from the multiselect product attribute that you created in the previous exercise.

• Update attributes and set frontend_model and is_html_allowed_on_front attribute properties.
• Create a frontend model class that renders attribute values as an HTML list rather than as comma-separated values.
• Run appropriate terminal commands to apply the upgrade
• Verify it works on a product view page.

**Solution**

1. Write update attribute data patch.

```php
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit4\MultiSelect\Setup\Patch\Data;

use Magento\Framework\Setup\ModuleDataSetupInterface;
use Magento\Framework\Setup\Patch\DataPatchInterface;
use Magento\Framework\Setup\Patch\PatchInterface;

use Magento\Catalog\Model\Product;
use Magento\Catalog\Setup\CategorySetupFactory;
use Magento\Catalog\Model\ResourceModel\Eav\Attribute as CatalogAttribute;


/**
 * Class CategoryAttrUpdate
 * @package Unit4\MultiSelect\Setup\Patch\Data
 */
class CategoryAttrUpdate implements DataPatchInterface
{
    /**
     * @var CategorySetupFactory
     */
    protected $categorySetupFactory;

    /**
     * @var ModuleDataSetupInterface
     */
    protected $moduleDataSetup;

    /**
     * CategoryAttrUpdate constructor.
     * @param CategorySetupFactory $categorySetupFactory
     * @param ModuleDataSetupInterface $moduleDataSetup
     */
    public function __construct(
```

```
        CategorySetupFactory $categorySetupFactory,
        ModuleDataSetupInterface $moduleDataSetup
    ) {
        $this->categorySetupFactory = $categorySetupFactory;
        $this->moduleDataSetup = $moduleDataSetup;
    }

    /**
     * @return DataPatchInterface|void
     */
    public function apply()
    {
        $this->moduleDataSetup->startSetup();

        $catalogSetup = $this->categorySetupFactory->create(['setup' => $this->moduleDataSetup]);
        $attrParams = [
            'frontend' => 'Unit4\MultiSelect\Model\Entity\Attribute\Frontend\HtmlList',
            'is_html_allowed_on_front' => 1
        ];
        $catalogSetup->updateAttribute(Product::ENTITY, 'custom_multiselect', $attrParams);

        $this->moduleDataSetup->endSetup();
    }

    /**
     * @return array|string[]
     */
    public static function getDependencies()
    {
        return [];
    }

    /**
     * @return array|string[]
     */
    public function getAliases()
    {
        return [];
    }
}
```

## 4.5.5. Create a select attribute with a predefined list of options.

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

**Solution**

1. Add new customer attribute by data patch.

```php
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit4\CustomPriority\Setup\Patch\Data;

use Magento\Customer\Setup\CustomerSetup;
use Magento\Framework\Setup\ModuleDataSetupInterface;

use Magento\Customer\Model\Customer;
use Magento\Customer\Setup\CustomerSetupFactory;
use Magento\Framework\Setup\Patch\PatchInterface;


/**
 * @package Unit4\CustomPriority\Setup\Patch\Data
 */
class CustomerAttr implements \Magento\Framework\Setup\Patch\DataPatchInterface
{
    /**
     * @var CustomerSetupFactory
     */
    protected $customerSetupFactory;

    /**
     * @var \Magento\Framework\Setup\ModuleDataSetupInterface
     */
    private $moduleDataSetup;

    /**
     * CustomerAttr constructor.
     * @param CustomerSetupFactory $customerSetupFactory
     * @param ModuleDataSetupInterface $moduleDataSetup
     */
    public function __construct(
        CustomerSetupFactory $customerSetupFactory,
        ModuleDataSetupInterface $moduleDataSetup
    ) {
        $this->customerSetupFactory = $customerSetupFactory;
        $this->moduleDataSetup = $moduleDataSetup;
    }

    /**
     * @return \Magento\Framework\Setup\Patch\DataPatchInterface|void
     */
    public function apply()
    {
        $this->moduleDataSetup->startSetup();

        /** @var CustomerSetup $customerSetup */
        $customerSetup = $this->customerSetupFactory->create(['setup' => $this->moduleDataSetup]);
```

```php
            $customerSetup->addAttribute(
                Customer::ENTITY,
                'custom_priority',
                [
                    'label' => 'Priority',
                    'type' => 'int',
                    'input' => 'select',
                    'source' =>
'\Unit4\CustomPriority\Model\Entity\Attribute\Frontend\Source\CustomerPriority',
                    'required' => 0,
                    'system' => 0,
                    'position' => 100,
                ]
            );
            $customerSetup->getEavConfig()
                ->getAttribute('customer', 'custom_priority')
                ->setData('used_in_forms', ['adminhtml_customer'])
                ->save();

            $this->moduleDataSetup->endSetup();
        }

        /**
         * @return array|string[]
         */
        public static function getDependencies()
        {
            return [];
        }

        /**
         * @return array|string[]
         */
        public function getAliases()
        {
            return [];
        }

    }
```

2. Create source model.

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

namespace Unit4\CustomPriority\Model\Entity\Attribute\Frontend\Source;
use Magento\Eav\Model\Entity\Attribute\Source\AbstractSource;

/**
 * @package Training\Orm\Entity\Attribute\Source
 */
```

```php
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;
    }
}
```

# Unit 5. Service Contracts

## Module 4. Services API: Repositories & Business Logic

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

### 5.4.1. (module name: Unit5_ProductList) 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
<?php
namespace Unit5\ProductList\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.

```
/**
 * Product constructor.
 * @param Context $context
 * @param ProductRepositoryInterface $productRepository
 * @param SearchCriteriaBuilder $searchCriteriaBuilder
 * @param FilterBuilder $filterBuilder
 * @param SortOrderBuilder $sortOrderBuilder
 */
public function __construct(
    Context $context,
    ProductRepositoryInterface $productRepository,
    SearchCriteriaBuilder $searchCriteriaBuilder,
    FilterBuilder $filterBuilder,
    SortOrderBuilder $sortOrderBuilder
) {
    parent::__construct($context);
    $this->productRepository = $productRepository;
    $this->searchCriteriaBuilder = $searchCriteriaBuilder;
    $this->filterBuilder = $filterBuilder;
    $this->sortOrderBuilder = $sortOrderBuilder;
}

// ... 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 must 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();
    $this->searchCriteriaBuilder->addFilters($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(SortOrder::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 two new lines at the end of the method.

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

### 5.4.2. (module name: Unit5_ CustomerList) 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`.
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.

```php
<?php

namespace Unit5\CustomerList\Controller\Repository;

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

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 = $this->getCustomersFromRepository();
        if(!empty($customers)) {
```

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

            $result = $this->resultFactory-
>create(\Magento\Framework\Controller\ResultFactory::TYPE_RAW);
            $result->setContents('Hello World!');
        }
        foreach ($customers as $customer) {
            $this->outputCustomer($customer);
        }
    }

    private function addEmailFilter()
    {
        $emailFilter = $this->filterBuilder
            ->setField('email')
            ->setValue('%@example.com')
            ->setConditionType('like')
            ->create();
        $this->filterGroupBuilder->addFilter($emailFilter);
    }
    private function addNameFilter()
    {
        $nameFilter = $this->filterBuilder
            ->setField('firstname')
            ->setValue('Veronica')
            ->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. (module name: Unit5_Repository) 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 solution code archive for details. The model, resource model, and collection code is boilerplate and is not included in this document.

2. Use a declarative schema to create the matching table, and a data patches class to create a couple of example records in the table.

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Setup/Declaration/Schema/etc/sch
ema.xsd">
  <table name="training_repository_example" resource="default" engine="innodb"
comment="training_repository_example">
    <column xsi:type="int" name="example_id" padding="10" unsigned="true"
nullable="false" identity="true" comment="Example_id"/>
    <column xsi:type="varchar" name="name" nullable="false" length="255"
comment="Name"/>
    <column xsi:type="timestamp" name="created_at" on_update="false" nullable="false"
default="CURRENT_TIMESTAMP" comment="Created_at"/>
    <column xsi:type="timestamp" name="updated_at" on_update="false" nullable="false"
default="CURRENT_TIMESTAMP" comment="Updated_at"/>
    <constraint xsi:type="primary" referenceId="PRIMARY">
      <column name="example_id"/>
    </constraint>
    <constraint xsi:type="unique" referenceId="TRAINING_REPOSITORY_EXAMPLE_NAME">
      <column name="name"/>
    </constraint>
  </table>
</schema>
```

```php
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit5\Repository\Setup\Patch\Data;

use Magento\Framework\Setup\Patch\DataPatchInterface;
use Magento\Eav\Setup\EavSetupFactory;
```

```php
use Magento\Framework\Setup\ModuleDataSetupInterface;

class AddTableData implements DataPatchInterface
{
    private $moduleDataSetup;

    public function __construct(
        ModuleDataSetupInterface $moduleDataSetup
    ) {
        $this->moduleDataSetup = $moduleDataSetup;
    }

    public function apply()
    {
        $setup = $this->moduleDataSetup;

        $setup->getConnection()->insertMultiple(
            $setup->getTable('training_repository_example'),
            [
                ['name' => 'Foo'],
                ['name' => 'Bar'],
                ['name' => 'Baz'],
                ['name' => 'Qux'],
            ]
        );
    }

    /**
     * getAliases
     *
     * @return void
     */
    public function getAliases()
    {
        return [];
    }

    /**
     * getDependencies
     *
     * @return void
     */
    public static function getDependencies()
    {
        return [];
    }
}
```

3.  Add an interface `Unit5\Repository\Api\ExampleRepositoryInterface`. It contains only the `getList()` method. No framework interface needs to be extended.

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

```
 */
namespace Unit5\Repository\Api;

use Magento\Framework\Api\SearchCriteriaInterface;

/**
 * Interface ExampleRepositoryInterface
 * @package Unit5\Repository\Api
 */
interface ExampleRepositoryInterface
{
    /**
     * @return Data\ExampleSearchResultsInterface
     */
    public function getList(SearchCriteriaInterface $searchCriteria);
}
```

4. Add an interface for the API data model `Unit5\Repository\Api\Data\ExampleInterface` with getters and setters for all the properties that should be accessible from the outside.

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit5\Repository\Api\Data;

/**
 * Interface ExampleInterface
 * @package Unit5\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 Unit5\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
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit5\Repository\Api\Data;

/**
 * Interface ExampleSearchResultsInterface
 * @package Unit5\Repository\Api\Data
 */
interface ExampleSearchResultsInterface extends
\Magento\Framework\Api\SearchResultsInterface
{
    /**
     * @api
     * @return \Unit5\Repository\Api\Data\ExampleInterface[]
     */
    public function getItems();

    /**
     * @api
     * @param \Unit5\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
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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/ObjectMa
nager/etc/config.xsd">
    <preference for="Unit5\Repository\Api\ExampleRepositoryInterface"
                type="Unit5\Repository\Model\ExampleRepository"/>


    <preference for="Unit5\Repository\Api\Data\ExampleInterface"
                type="Unit5\Repository\Model\Example"/>


    <preference for="Unit5\Repository\Api\Data\ExampleSearchResultsInterface"
                type="Magento\Framework\Api\SearchResults"/>
</config>
```

7. Make the `Unit5\Repository\Model\Example` class implement the `Api\Data\ExampleInterface`. Make the `Unit5\Repository\Model\Resource` class and `Collection` class inside `Unit5\Repository\Model\Resource\Example`.

```php
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit5\Repository\Model;

use Magento\Framework\Model\AbstractModel;
use Unit5\Repository\Api\Data\ExampleInterface;

/**
 * Class Example
 * @package Unit5\Repository\Model
 */
class Example extends AbstractModel implements ExampleInterface
{
    /**
     *
     */
    protected function _construct()
    {
        $this->_init(Resource\Example::class);
    }

    /**
     * @return mixed
     */
    public function getName()
    {
        return $this->_getData('name');
    }

    /**
     * @param string $name
     */
    public function setName($name)
    {
```

```php
        $this->setData('name', $name);
    }

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

    /**
     * @param string $createdAt
     */
    public function setCreatedAt($createdAt)
    {
        $this->setData('modified_at', $createdAt);
    }

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

    /**
     * @param string $modifiedAt
     */
    public function setModifiedAt($modifiedAt)
    {
        $this->setData('modified_at', $modifiedAt);
    }
}

<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit5\Repository\Model\Resource;

/**
 * Class Example
 * @package Unit5\Repository\Model\Resource
 */
class Example extends \Magento\Framework\Model\ResourceModel\Db\AbstractDb
{
    /**
     *
     */
    protected function _construct()
    {
        $this->_init('training_repository_example', 'example_id');
```

```php
    }
}

<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit5\Repository\Model\Resource\Example;

/**
 * Class Collection
 * @package Unit5\Repository\Model\Resource\Example
 */
class Collection extends
\Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection implements
\Magento\Framework\Api\Search\SearchResultInterface
{
    /**
     * @var $aggregations
     */
    protected $aggregations;

    /**
     *
     */
    protected function _construct()
    {
        $this->_init('Unit5\Repository\Model\Example',
'Unit5\Repository\Model\Resource\Example');
    }

    /**
     * @return AggregationInterface
     */
    public function getAggregations()
    {
        return $this->aggregations;
    }

    /**
     * @param AggregationInterface $aggregations
     * @return $this
     */
    public function setAggregations($aggregations)
    {
        $this->aggregations = $aggregations;
    }

    /**
     * Get search criteria.
     *
     * @return \Magento\Framework\Api\SearchCriteriaInterface|null
     */
    public function getSearchCriteria()
    {
```

```
        return null;
    }

    /**
     * Set search criteria.
     *
     * @param \Magento\Framework\Api\SearchCriteriaInterface $searchCriteria
     * @return $this
     * @SuppressWarnings(PHPMD.UnusedFormalParameter)
     */
    public function setSearchCriteria(\Magento\Framework\Api\SearchCriteriaInterface
$searchCriteria = null)
    {
        return $this;
    }

    /**
     * Get total count.
     *
     * @return int
     */
    public function getTotalCount()
    {
        return $this->getSize();
    }

    /**
     * Set total count.
     *
     * @param int $totalCount
     * @return $this
     * @SuppressWarnings(PHPMD.UnusedFormalParameter)
     */
    public function setTotalCount($totalCount)
    {
        return $this;
    }


    /**
     * Set items list.
     *
     * @param \Magento\Framework\Api\ExtensibleDataInterface[] $items
     * @return $this
     * @SuppressWarnings(PHPMD.UnusedFormalParameter)
     */
    public function setItems(array $items = null)
    {
        return $this;
    }
}
```

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
<?php
/**
 *
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit5\Repository\Model;

use Magento\Framework\Api\Search\FilterGroup;
use Magento\Framework\Api\SearchCriteriaInterface;
use Unit5\Repository\Api\Data\ExampleInterface;
use Unit5\Repository\Api\Data\ExampleInterfaceFactory as ExampleDataFactory;
use Unit5\Repository\Api\Data\ExampleSearchResultsInterface;
use Unit5\Repository\Api\Data\ExampleSearchResultsInterfaceFactory;
use Unit5\Repository\Api\ExampleRepositoryInterface;
use Unit5\Repository\Model\Example as ExampleModel;
use Unit5\Repository\Model\Resource\Example\Collection as ExampleCollection;
use Magento\Framework\Api\SortOrder;

/**
 * Class ExampleRepository
 * @package Unit5\Repository\Model
 */
class ExampleRepository implements ExampleRepositoryInterface
{
    /**
     * @var ExampleSearchResultsInterfaceFactory
     */
    private $searchResultsFactory;
    /**
     * @var ExampleFactory
     */
    private $exampleFactory;
    /**
     * @var ExampleDataFactory
     */
    private $exampleDataFactory;

    /**
     * ExampleRepository constructor.
     * @param ExampleSearchResultsInterfaceFactory $searchResultsFactory
     * @param ExampleFactory $exampleFactory
     * @param ExampleDataFactory $exampleDataFactory
     */
    public function __construct(
        ExampleSearchResultsInterfaceFactory $searchResultsFactory,
        ExampleFactory $exampleFactory,
        ExampleDataFactory $exampleDataFactory
    ) {
```

```php
            $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;
        }

        /**
         * @param FilterGroup $filterGroup
         * @param ExampleCollection $collection
         */
        private function addFilterGroupToCollection(
            FilterGroup $filterGroup,
            ExampleCollection $collection
        ) {
            $fields = [];
            $conditions = [];
            foreach ($filterGroup->getFilters() as $filters) {
                foreach ($filters as $filter){
                    $condition = $filter->getConditionType() ? $filter-
>getConditionType() : 'eq';
                    $fields[] = $filter->getField();
                    $conditions[] = [$condition => $filter->getValue()];
                }
            }
            if ($fields) {
                $collection->addFieldToFilter($fields, $conditions);
            }
        }

        /**
         * @param ExampleCollection $collection
         * @return array
         */
        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;
    }

    /**
     * @param SearchCriteriaInterface $searchCriteria
     * @param ExampleCollection $collection
     */
    private function applySearchCriteriaToCollection(
        SearchCriteriaInterface $searchCriteria,
        ExampleCollection $collection
    ) {
        $this->applySearchCriteriaFiltersToCollection(
            $searchCriteria,
            $collection
        );
        $this->applySearchCriteriaSortOrdersToCollection(
            $searchCriteria,
            $collection
        );
        $this->applySearchCriteriaPagingToCollection(
            $searchCriteria,
            $collection
        );
    }

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

    /**
     * @param SearchCriteriaInterface $searchCriteria
     * @param ExampleCollection $collection
     */
    private function applySearchCriteriaSortOrdersToCollection(
        SearchCriteriaInterface $searchCriteria,
        ExampleCollection $collection
    ) {
        $sortOrders = $searchCriteria->getSortOrders();

        if ($sortOrders) {
            foreach ($sortOrders as $sortOrder) {
                $isAscending = $sortOrder->getDirection() == SortOrder::SORT_ASC;
                $collection->addOrder(
```

```
                    $sortOrder->getField(),
                    $isAscending ? 'ASC' : 'DESC'
                );
            }
        }
    }

    /**
     * @param SearchCriteriaInterface $searchCriteria
     * @param ExampleCollection $collection
     */
    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
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit5\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 Unit5\Repository\Api\ExampleRepositoryInterface;

/**
 * Class Example
 * @package Unit5\Repository\Controller\Repository
 */
class Example extends Action
{
    /**
     * @var ExampleRepositoryInterface
     */
    private $exampleRepository;
    /**
     * @var SearchCriteriaBuilder
     */
    private $searchCriteriaBuilder;
    /**
     * @var FilterBuilder
     */
    private $filterBuilder;

    /**
     * Example constructor.
```

```php
     * @param Context $context
     * @param ExampleRepositoryInterface $exampleRepository
     * @param SearchCriteriaBuilder $searchCriteriaBuilder
     * @param FilterBuilder $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']);
        $this->searchCriteriaBuilder->addFilters($filters);
        $examples = $this->exampleRepository->getList(
            $this->searchCriteriaBuilder->create()
        )->getItems();
        foreach ($examples as $example) {
            $this->getResponse()->appendBody(sprintf(
                "%s (%d)\n",
                $example->getName(),
                $example->getId()
            ));
        }
    }
}
```

# Unit 6. AdminHTML

## Module 2. Adminhtml: System Configuration – Menu – ACL

### 6.2.1. (solution module Unit6_SystemConfiguration) Add Element to the System Configuration

- Create a new element in the system configuration.

    - Name it "test".
    - Put it in the General section.
    - Make it a "yes/no" selection.

- 6.6.1b: Create a new element in the system configuration with custom code that renders "Hello World".

**Solution**

1. `system.xml` file can be found here `etc/adminhtml`.

```xml
<?xml version="1.0"?>
<!--
/**
* Copyright © Magento. All rights reserved.
* See COPYING.txt for license details.
*/
-->
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Config:etc/system_file.xsd">
    <system>
        <section id="general">
            <group id="Unit6" type="text" translate="label" sortOrder="100"
                    showInDefault="1" showInStore="1" showInWebsite="1">
                <label>Unit6 Fields</label>
                <field id="test" type="select" translate="label comment" sortOrder="7"
                        showInWebsite="0" showInStore="0" showInDefault="1">
                    <label>Test Field</label>
                    <comment>This is test select field</comment>
                    <source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
                </field>
                <field id="custom" type="select" translate="label comment" sortOrder="10"
                        showInDefault="1" showInStore="0" showInWebsite="0">
                    <label>Custom test field</label>
                    <comment>This's another test field</comment>

<frontend_model>Unit6\SystemConfiguration\Block\Config\Custom</frontend_model>
                </field>
            </group>
        </section>
    </system>
</config>
```

2. Create a block which renders configuration element.

```php
<?php
/**
 *  Copyright © Magento. All rights reserved.
```

```
 *  See COPYING.txt for license details.
 */
namespace Unit6\SystemConfiguration\Block\Config;

use Magento\Config\Block\System\Config\Form\Field;
use Magento\Framework\Data\Form\Element\AbstractElement;


/**
 * Class Custom
 * @package Unit6\SystemConfiguration\Block\Config
 */
class Custom extends Field
{
    /**
     * @param AbstractElement $element
     * @return string
     */
    protected function _getElementHtml(AbstractElement $element)
    {
        return 'This is custom config block output';
    }
}
```

## 6.2.2. (solution module Unit6_AdminMenu). Admin Menu New Element

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


**Solution**
1. Create `menu.xml` in the adminhtml area.

```xml
<?xml version="1.0"?>
<!--
/**
* Copyright © Magento. All rights reserved.
* See COPYING.txt for license details.
*/
-->
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Backend:etc/menu.xsd">
    <menu>
        <add id="Unit6_AdminMenu::new_menu_item" title="New Menu Item"
module="Unit6_AdminMenu"
             resource="Unit6_AdminPage::new_page" parent="Magento_Catalog::inventory"
             action="adminpage/page" sortOrder="199"/>
    </menu>
</config>
```


## 6.2.3. (solution module Unit6_AdminPage) Create New ACL Resource

- 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**

1. Create an `acl.xml` in the `etc` folder. Add a new ACL resource.

```xml
<?xml version="1.0"?>
<!--
  ~  Copyright © 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:Acl/etc/acl.xsd">
    <acl>
        <resources>
            <resource id="Magento_Backend::admin">
                        <resource id="Unit6_AdminPage::new_page" title="New Admin Page"
                                 translate="title"  sortOrder="100"/>
            </resource>
        </resources>
    </acl>
</config>
```

2. Add new menu item at etc/adminhtml/menu.xml

```xml
<?xml version="1.0"?>
<!--
/**
* Copyright © Magento. All rights reserved.
* See COPYING.txt for license details.
*/
-->
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Backend:etc/menu.xsd">
    <menu>
        <add id="Unit6_AdminPage::new_admin_page" title="New Admin Page"
module="Unit6_AdminPage"
              resource="Unit6_AdminPage::new_page" parent="Magento_Catalog::inventory"
              action="adminpage/page/index" sortOrder="10"/>
    </menu>
</config>
```

3. Create admin route.

```xml
<?xml version="1.0"?>
<!--
/**
* Copyright © 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="adminpage" frontName="adminpage">
            <module name="Unit6_AdminPage"/>
        </route>
    </router>
</config>
```

4. Make admin action class with ACL resource restriction.

```php
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit6\AdminPage\Controller\Adminhtml\Page;

use Magento\Backend\App\Action\Context;
use Magento\Framework\View\Result\PageFactory;

/**
 * Class Index
 * @package Unit6\AdminPage\Controller\Adminhtml\Index
 */
class Index extends \Magento\Backend\App\Action
{
    const ADMIN_RESOURCE = "Unit6_AdminPage::new_page";

    /**
     * @var \Magento\Framework\View\Result\PageFactory;
     */
    protected $resultPageFactory;

    /**
     * Index constructor.
     * @param PageFactory $resultPageFactory
     */
    public function __construct(PageFactory $resultPageFactory, Context $context)
    {
        $this->resultPageFactory = $resultPageFactory;
        parent::__construct($context);
    }

    /**
     * @return \Magento\Framework\View\Result\Page
     */
    public function execute()
    {
        /** @var \Magento\Backend\Model\View\Result\Page $backendPage */
        $backendPage = $this->resultPageFactory->create();
        $backendPage->setActiveMenu('Unit6_AdminPage::new_admin_page');
        $backendPage->addBreadcrumb(__('Dashboard'),__('New Admin Page'));
        $backendPage->getConfig()->getTitle()->set('New Admin Page');

        return $backendPage;
    }
}
```

5. Create admin page layout and template.

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © Magento. All rights reserved.
```

```
* See COPYING.txt for license details.
*/
-->
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configuration.xsd"
>
    <body>
        <referenceContainer name="content">
            <block class="Magento\Framework\View\Element\Template" name="new.page.content"
                    template="Unit6_AdminPage::index.phtml"/>
        </referenceContainer>
    </body>
</page>
```

**Template phtml file:**

```
<?= "Hey admin, this is the Magento developer ../" ?>
```

6. Now, let's go to the admin. Create new user role, check our ACL resource is included. Re-sign in. Go to the page.