Magento U
An Adobe Company

Exercises & Solutions

Fundamentals of Magento 2 Development Part 2

https://u.magento.com/

v2.3

Magento® U

training@magento.com

u.magento.com

u.magento.com/certification

Contents

Unit 1. Ui Components

1.2 Architecture and Configuration

1.2.2.1

Write a module/class that logs the data that is supplied to the customer add/edit form before it is rendered in the Admin panel. This data can be printed as an array into a log file.

Solution

1. Create a folder app/code/Unit1/CustomerFormDump.

   Create a file app/code/Unit1/CustomerFormDump/etc/module.xml:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright Â© 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_CustomerFormDump">
    </module>
</config>
```

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

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

\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'Unit1_CustomerFormDump',
    __DIR__
);
```

3. To add a plugin, use the following code in di.xml:

```
<?xml version="1.0"?>
```

```
<!--
/**
 * Copyright Â© 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\Customer\Model\Customer\DataProviderWithDefaultAddresses">
        <plugin name="formLog" type="Unit1\CustomerFormDump\Plugin\Log" />
    </type>
</config>
```

Now you can implement your plugin:

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

class Log
{

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

    public function __construct(\Psr\Log\LoggerInterface $logger)
    {
        $this->logger = $logger;
    }
```

```
    /**
     * afterGetData
     *
     * @param  mixed $subject
     * @param  mixed $result
     *
     * @return void
     */
    public function
afterGetData(\Magento\Customer\Model\Customer\DataProviderWithDefaultAddresses $subject,
$result)
    {
        foreach($result as $customer){
            $this->logger->debug('data: ', $customer);
        }

        return $result;
    }
}
```

4. Enable your module. Run `php bin/magento module:enable Unit1_CustomerFormDump`
5. Run `bin/magento setup:upgrade` from the Magento root directory to upgrade your database.

1.3 Templates and Rendering

1.3.2.1

Create an extension that writes “Hello World” at the top of a mini-cart popup.

Solution

1. Create a default.xml layout file in your module
2. Reference the minicart block.
3. Add new item UiComponent to minicart with HTML template

```
<?xml version="1.0"?>
<!--
/**
```

```
 * Copyright Â© 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="minicart">
            <arguments>
                <argument name="jsLayout" xsi:type="array">
                    <item name="components" xsi:type="array">
                        <item name="minicart_content" xsi:type="array">
                            <item name="children" xsi:type="array">
                                <item name="subtotal.container" xsi:type="array">
                                    <item name="children" xsi:type="array">
                                        <item name="hello_info" xsi:type="array">
                                            <item name="component"
xsi:type="string">uiComponent</item>
                                            <item name="config" xsi:type="array">
                                                <item name="displayArea"
xsi:type="string">helloInfo</item>
                                            </item>
                                            <item name="children" xsi:type="array">
                                                <item name="hello" xsi:type="array">
                                                    <item name="component"
xsi:type="string">uiComponent</item>
                                                    <item name="config" xsi:type="array">
                                                        <item name="template"
xsi:type="string">Unit1_HelloMiniCart/checkout/minicart/hello</item>
                                                    </item>
                                                </item>
                                            </item>
                                        </item>
```

```
                                </item>
                            </item>
                        </item>
                    </item>
                </argument>
            </arguments>
        </referenceBlock>
    </body>
</page>
```

4. Create HTML template in the following folder:

   Unit1\HelloMiniCart\view\frontend\web\template\checkout\minicart\hello.html

```
<!--
/**
 * Copyright Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<span class="label" translate="'Hello World'" />
```

1.3.2.2

Create an HTML template for a previous exercise that renders a list of customers from DataProvider (list of customers could be hardcoded) into an HTML template. A skeleton module – Unit1/ StandaloneXhtmlTemplateSkeleton has been provided – build on top of it to complete the exercise. It has a pre-defined DataProvider.

Solution

5. Create a folder app/code/Unit1/StandaloneXhtmlTemplateSkeleton.

   Create a file app/code/Unit1/StandaloneXhtmlTemplateSkeleton/etc/module.xml:

```
<?xml version="1.0"?>
<!--
  ~  Copyright Â© 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_StandaloneXhtmlTemplateSkeleton"/>
</config>
```

6. Register your module with app/code/Unit1/StandaloneXhtmlTemplateSkeleton/registration.php:

```
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
use Magento\Framework\Component\ComponentRegistrar;

ComponentRegistrar::register(
    ComponentRegistrar::MODULE,
    'Unit1_StandaloneXhtmlTemplateSkeleton',
    __DIR__
);
```

7. Put your data provider class in the Model\DataProvider.php:

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

class DataProvider extends \Magento\Ui\DataProvider\AbstractDataProvider
{
    /**
     * getData
     *
```

```
     * @return void
     */
    public function getData()
    {
        return [ 'list' => [
            0 =>
                [
                    'name'      => 'Veronica',
                    'lastname'  => 'Costello'
                ]
            ]
        ];
    }
}
```

1.3.2.3

Create a standalone component that renders hardcoded text from default.xhtml template. Make that phrase available for all pages.

Solution

1. Create a default.xml layout file in your module:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright Â© 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.x
sd">
    <body>
        <referenceContainer name="content">
            <uiComponent name="unit1_grid_listing_list"/>
```

```
        </referenceContainer>
    </body>
</page>
```

2. Create a default.xhtml template:

```
<?xml version="1.0" encoding="UTF-8"?>
<div>
    <p>Hardcoded Text</p>

    <div data-bind="scope: 'unit1_grid_listing_list.unit1_grid_listing_list'"
class="entry-edit form-inline"></div>
</div>
```

1.4 JavaScript Role in UiComponents

1.4.2.1

Create a simple UiComponent which will render the “Hello World” phrase on the page.

Solution

1. Create the backend action:

```
<?php
namespace Unit1\SimpleUiComponent\Controller\Adminhtml\UiComponent;

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

class Index extends Action
{
    /**
     * ACL access restriction
     */
    const ADMIN_RESOURCE = 'Unit1_SimpleUiComponent::simple_list';
```

```
    /**
     * View page action
     *
     * @return \Magento\Framework\Controller\ResultInterface
     */
    public function execute()
    {
        return $this->resultFactory->create(ResultFactory::TYPE_PAGE);
    }
}
```

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

```
<?xml version="1.0"?>
<!--
/**
* Copyright Â© 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="simple" frontName="simple">
            <module name="Unit1_SimpleUiComponent" before="Magento_Backend"/>
        </route>
    </router>
</config>
```

3. Create menu.xml in the adminhtml area:

```
<?xml version="1.0"?>
<!--
/**
* Copyright Â© 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="Unit1_SimpleUiComponent::simple_list" title="Simple UiComponent"
module="Unit1_SimpleUiComponent"
             parent="Magento_Catalog::catalog"
resource="Unit1_SimpleUiComponent::grid"
             sortOrder="0" action="simple/uicomponent/index"/>
    </menu>
</config>
```

4. Create a simple_uicomponent_index.xml layout file in your module
5. Reference the content container.
6. Add new block and specify the template in the “template” attribute

```
<?xml version="1.0"?>
<!--
/**
 * Copyright Â© 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="content">
            <block name="simple_uicomponent"
template="Unit1_SimpleUiComponent::index.phtml" />
        </referenceContainer>
```

```
    </body>
</page>
```

7. Create PHTML template in the following folder:

   Unit1\SimpleUiComponent\view\adminhtml\templates\index.phtml

```
<div data-bind="scope: 'simple-component'">
    <div data-bind="text: simple"></div>
</div>
<script type="text/x-magento-init">
   {
       "*": {
           "Magento_Ui/js/core/app": {
               "components": {
                   "simple-component": {
                       "component": "Unit1_SimpleUiComponent/js/module"
                   }
               }
           }
       }
   }
</script>
```

1.4.2.2

Use Unit1/JsDataProviderSkeleton. It has a pre-defined DataProvider.

Use the generic URL to fetch data for your component – the URL should use your DataProvider fetch the data. Render the data to the HTML template.

Solution

1. Create a folder app/code/Unit1/JsDataProviderSkeleton.

   Create a file app/code/Unit1/JsDataProviderSkeleton/etc/module.xml:

```
<?xml version="1.0"?>
<!--
  ~  Copyright Â© 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_JsDataProviderSkeleton"/>
</config>
```

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

```
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
use Magento\Framework\Component\ComponentRegistrar;

ComponentRegistrar::register(
    ComponentRegistrar::MODULE,
    'Unit1_JsDataProviderSkeleton',
    __DIR__
);
```

3. Put your data provider class in the Mode\DataProvider.php:

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

class DataProvider extends \Magento\Ui\DataProvider\AbstractDataProvider
{
    /**
     * getData
     *
```

```
     * @return void
     */
    public function getData()
    {
        return [ 'list' => [
            0 =>
                [
                    'name'      => 'Veronica',
                    'lastname'  => 'Costello'
                ]
            ]
        ];
    }
}
```

4. Create a default.xml layout file in your module and add a grid UiComponent to the content container:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright Â© Magento, Inc. 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>
        <referenceContainer name="content">
            <uiComponent name="unit1_grid_listing"/>
        </referenceContainer>
    </body>
</page>
```

5. Now we specify the configuration of the `unit1_grid_listing` UiComponent, and it's not the easiest part of the task:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
/**
 * Copyright Â© Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<listing xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configuration.xsd">

    <!-- This technique is only for educational purpose! Donâ€™t use it in production
code! -->
    <htmlContent name="unit1">
        <argument name="block"
xsi:type="object">Magento\Framework\View\Element\Template</argument>
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="provider"
xsi:type="string">unit1_grid_listing.unit1_grid_listing_data_source</item>
                <item name="component"
xsi:type="string">Unit1_JsDataProviderSkeleton/js/data</item>
                <item name="template"
xsi:type="string">Unit1_JsDataProviderSkeleton/index</item>
            </item>
        </argument>
    </htmlContent>
    <!-- This technique is only for educational purpose! Donâ€™t use it in production
code! -->

    <dataSource name="unit1_grid_listing_data_source"
component="Magento_Ui/js/grid/provider">
        <settings>
            <updateUrl path="mui/index/render"/>
        </settings>
```

```
        <dataProvider class="Unit1\JsDataProviderSkeleton\Model\DataProvider"
name="unit1_grid_listing_data_source">
            <settings>
                <requestFieldName>id</requestFieldName>
                <primaryFieldName>grid_id</primaryFieldName>
            </settings>
        </dataProvider>
    </dataSource>
</listing>
```

6. Create new JavaScript component:
   Unit1\JsDataProviderSkeleton\view\frontend\web\js\data.js

```
/**
 * Copyright Â© Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
define(["uiComponent"], Component => {
    "use strict";

    return Component.extend({
        defaults: {
            imports: {
                // Code here
            }
        },

    // Code here
    });
});
```

7. Create HTML template:
   Unit1\JsDataProviderSkeleton\view\frontend\web\template\index.html

Unit 2. Introductions to Grids and Forms

2.1 Introduction to Grids

2.1.4.1

A merchant would like to add the ability to verify orders. They also want to be able to view if an order has been verified or not from the sales order grid.

How would you accomplish this? Add a require_verification column to the sales order table and the order grid.

Solution

1. To add a new column to the order table, create a `db_schema.xml`:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright Â© Magento, Inc. 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/schema.
xsd">
    <table name="sales_order" resource="sales" engine="innodb" comment="Sales Flat
Order">
        <column xsi:type="smallint" name="require_verification" padding="5"
unsigned="true" nullable="true" identity="false" comment="Require Verification"/>
    </table>
    <table name="sales_order_grid" resource="sales" engine="innodb" comment="Sales Flat
Order Grid">
        <column xsi:type="smallint" name="require_verification" padding="5"
unsigned="true" nullable="true" identity="false" comment="Require Verification"/>
    </table>
</schema>
```

2. Create order listing configuration XML file – `sales_order_grid.xml`:

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

xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configuration.xsd">
     <columns name="sales_order_columns">
        <column name="require_verification">
            <settings>
                <filter>select</filter>
                <dataType>select</dataType>
                <label translate="true">Require Verification</label>
            </settings>
        </column>
    </columns>
</listing>
```

3. And finally, create a `di.xml` file:

```
<?xml version="1.0"?>
<!--
/**
* Copyright Â© 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">
    <virtualType name="Magento\Sales\Model\ResourceModel\Order\Grid"
type="Magento\Sales\Model\ResourceModel\Grid">
```

```
        <arguments>
            <argument name="columns" xsi:type="array">
                <item name="require_verification"
xsi:type="string">sales_order.require_verification</item>
            </argument>
        </arguments>
    </virtualType>
</config>
```

2.1.5.1

Add a new filter for configurable products on a product grid page that filters by number of configurable options.

- **Create product_listing ui_component**
- **Create data provider class implements AddFilterToCollectionInterface**
- **Make an option provider class implements OptionSourceInterface**
- **And don’t forget to add configurable_options item to the Magento\Catalog\Ui\DataProvider\Product\ProductDataProvider addFiltersStrategies constructor parameter in the di.xml file**
- **Check that all it works**

Solution

1. Create a product_listing.xml file and put it in view\adminhtml\ui_component:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
/**
* Copyright Â© Magento. All rights reserved.
* See COPYING.txt for license details.
*/
-->
<listing xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configuration.xsd">
<listingToolbar name="listing_top">
    <filters name="listing_filters">
```

```
      <filterSelect name="configurable_options" sortOrder="100" provider="${ $.parentName
}">
          <settings>
              <options class="Unit2\ConfigurableProducts\Ui\Component\Listing\Options"/>
              <label translate="true">Configurable options</label>
              <dataScope>configurable_options</dataScope>
          </settings>
      </filterSelect>
    </filters>
</listingToolbar>
</listing>
```

2. Put your data provider class in the Ui\DataProvider\Product folder:

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

use
Magento\ConfigurableProduct\Model\ResourceModel\Product\Type\Configurable\Attribute\Colle
ction as AttributesCollection;
use Magento\Framework\Data\Collection;
use Magento\Ui\DataProvider\AddFilterToCollectionInterface;

/**
 * Class AddConfigurableOptionsToCollection
 * @package Unit2\ConfigurableProducts\Ui\DataProvider\Product
 */
class AddConfigurableOptionsToCollection implements AddFilterToCollectionInterface
{
    /**
     * @var AttributesCollection $attributeCollection
```

```
     */
    protected $attributeCollection = null;

    /**
     * AddConfigurableOptionsToCollection constructor.
     *
     * @param AttributesCollection $collection
     */
    public function __construct(AttributesCollection $collection)
    {
        $this->attributeCollection = $collection;
    }



    /**
     * @param Collection $collection
     * @param string $field
     * @param null $condition
     * @throws \Magento\Framework\Exception\LocalizedException
     */
    public function addFilter(Collection $collection, $field, $condition = null)
    {
        if (isset($condition['eq']) && ($numberOfOptions = $condition['eq'])) {
            $select = clone  $this->attributeCollection->getSelect();
            $select->reset(\Zend_Db_Select::COLUMNS)
                   ->columns(array('product_id', 'COUNT(*) as cnt'))
                   ->group('product_id');

            $res = $this->attributeCollection->getConnection()->fetchAll($select);

            $ids = [];
            foreach ($res as $opt) {
                if ($opt['cnt'] == $numberOfOptions) {
```

```
                    $ids[] = $opt['product_id'];
                }
            }
            $collection->addFieldToFilter('entity_id', ['in' => $ids]);
        }
    }

}
```

3. The option provider class has to be placed in the `Ui\Component\Listing` folder:

```
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\ConfigurableProducts\Ui\Component\Listing;

use Magento\Framework\Data\OptionSourceInterface;

/**
 * Class Options
 * @package Unit2\ConfigurableProducts\Ui\Component\Listing
 */
class Options implements OptionSourceInterface
{
    const ATTR_OPTIONS = [
        ['label' => 'Default', 'value' => ''],
        ['label' => '1',     'value' => '1'],
        ['label' => '2',     'value' => '2'],
        ['label' => '3',     'value' => '3']
    ];

    /**
```

```
     * @return array
     */
    public function toOptionArray()
    {
        return self::ATTR_OPTIONS;
    }
}
```

4. make a `di.xml` configuration file:

```
<?xml version="1.0"?>
<!--
/**
* Copyright Â© 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\Ui\DataProvider\Product\ProductDataProvider">
        <arguments>
            <argument name="addFilterStrategies" xsi:type="array">
                <item name="configurable_options"
xsi:type="object">Unit2\ConfigurableProducts\Ui\DataProvider\Product\AddConfigurableOptio
nsToCollection</item>
            </argument>
        </arguments>
    </type>
</config>
```

2.1.5.2

Add new attribute to the product type of varchar (call it product_series)

Create a filter on a product grid for that attribute.

Check that all it works.

Solution

1. Create a data patch that will add a `product_series` attribute to the product entity
2. Make a filter on the product grid based on this attribute

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

namespace Unit2\ProductSeries\Setup\Patch\Data;

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

/**
 * Class ProductSeriesAttr
 * @package Unit2\ProductSeries\Setup\Patch\Data
 */
class ProductSeriesAttr implements DataPatchInterface
{
    /**
     * @var EavSetup
     */
    protected $eavSetup;

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

```
    /**
     * ProductSeriesAttr constructor.
     * @param EavSetup $eavSetup
     * @param ModuleDataSetupInterface $moduleDataSetup
     */
    public function __construct(EavSetup $eavSetup, ModuleDataSetupInterface
$moduleDataSetup)
    {
        $this->eavSetup = $eavSetup;
        $this->moduleDataSetup = $moduleDataSetup;
    }

    /**
     * @return DataPatchInterface|void
     * @throws \Magento\Framework\Exception\LocalizedException
     */
    public function apply()
    {
        $entityTypeId = $this->eavSetup->getEntityTypeId('catalog_product');
        $attributeSetId = $this->eavSetup->getAttributeSetId($entityTypeId, 'Bag');
        $attributeGroupId = $this->eavSetup->getAttributeGroupId($entityTypeId,
$attributeSetId, 'Product Details');
        $attributeCode = 'product_series';
        $properties = [
            'type' => 'varchar',
            'label' => 'Product Series',
            'global' =>
\Magento\Eav\Model\Entity\Attribute\ScopedAttributeInterface::SCOPE_GLOBAL,
            'user_defined' => 1,
            'required' => 0,
            'visible_on_front' => 1,
            'is_used_in_grid' => 1,
            'is_visible_in_grid' => 1,
            'is_filterable_in_grid' => 1
```

```
        ];

        $this->eavSetup->addAttribute($entityTypeId, $attributeCode, $properties);
        $this->eavSetup->addAttributeToGroup($entityTypeId, $attributeSetId,
$attributeGroupId, $attributeCode);
    }

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

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

}
```

2.1.5.3

Create custom multi select filter for the product grid page. For example, filter by category or for existing attribute such as Attribute Set. A skeleton module – Unit2/CustomFilterSkeleton has been provided – build on top of it to complete the exercise. It has a pre-defined container for new custom filter type.

Solution

1. Create a file app/code/Unit1/CustomFilterSkeleton/etc/module.xml:

```
<?xml version="1.0"?>
```

```
<!--
/**
* Copyright Â© 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="Unit2_CustomFilterSkeleton"/>
</config>
```

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

```
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'Unit2_CustomFilterSkeleton',
    __DIR__
);
```

3. Create a product_listing.xml file and put it in view\adminhtml\ui_component:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
/**
 * Copyright Â© Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<listing xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configuration.xsd">
    <listingToolbar name="listing_top">
```

```
        <filters name="listing_filters"
component="Unit2_CustomFilterSkeleton/js/grid/columns/filters">
        </filters>
    </listingToolbar>

    <columns name="product_columns">
        <column name="attribute_set_id" sortOrder="50">
            <settings>
                <filter>multiSelect</filter>
                <dataType>multiselect</dataType>
                <addField>true</addField>
            </settings>
        </column>
    </columns>
</listing>
```

4. Create new JavaScript component:
   Unit2\CustomFilterSkeleton\view\adminhtml\web\js\grid\columns\filter.js

```
/**
 * Copyright © Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */

define([
    'Magento_Ui/js/grid/filters/filters'
], function (Filters) {
    'use strict';

    return Filters.extend({
        defaults: {
            templates: {
                filters: {
                    multiSelect: {
                        component: 'Magento_Ui/js/form/element/ui-select',
```

```
                        template: 'ui/grid/filters/elements/ui-select',
                        options: '${ JSON.stringify($.$data.column.options) }',
                        caption: ' '
                    }
                }
            }
        }
    });
});
```

2.1.6.1

Create a mass action to the orders grid which changes require_verification to “0” for all selected orders with status pending.

Create a route in adminhtml area.

Solution

1. Create a route in the `adminhtml` area:

```
<?xml version="1.0"?>
<!--
/**
* Copyright Â© 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="verification" frontName="verification">
            <module name="Unit2_RequireVerification" before="Magento_Backend"/>
        </route>
    </router>
</config>
```

2. Create a mass action class to the orders grid which changes it to “0” for all selected orders:

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

use Magento\Backend\App\Action;
use Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection;
use Magento\Sales\Controller\Adminhtml\Order\AbstractMassAction;
use Magento\Ui\Component\MassAction\Filter;
use Magento\Sales\Model\ResourceModel\Order\CollectionFactory;

/**
 * Class Verify
 * @package Unit2\RequireVerification\Controller\Adminhtml\Order
 */
class Verify extends AbstractMassAction
{
    /**
     * Verify constructor.
     *
     * @param Action\Context $context
     * @param Filter $filter
     * @param CollectionFactory $collection
     */
    public function __construct(Action\Context $context, Filter $filter,
CollectionFactory $collection)
    {
        parent::__construct($context, $filter);
        $this->collectionFactory = $collection;
    }
```

```
    /**
     * @param AbstractCollection $collection
     * @return \Magento\Framework\Controller\Result\Redirect
     */
    protected function massAction(AbstractCollection $collection)
    {
        foreach ($collection as $order) {
            if ($order->getStatus() == 'pending')
            {
                $order->setRequireVerification(0)->save();
            }
}
        $resultRedirect = $this->resultRedirectFactory->create();
        $resultRedirect->setPath($this->getComponentRefererUrl());

        return $resultRedirect;
    }
}
```

3. Add mass action to the grid - sales_order_grid.xml

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

xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configuration.xsd">
     <columns name="sales_order_columns">
        <column name="require_verification">
```

```
            <settings>
                <filter>select</filter>
                <dataType>select</dataType>
                <label translate="true">Require Verification</label>
            </settings>
        </column>
    </columns>
    <listingToolbar name="listing_top">
        <massaction name="listing_massaction" component="Magento_Ui/js/grid/tree-
massactions">
            <action name="verify">
                <settings>
                    <url path="verification/order/verify"/>
                    <type>verify</type>
                    <label translate="true">Verify</label>
                </settings>
            </action>
        </massaction>
    </listingToolbar>
</listing>
```

2.1.7.1

For the require_verification column in the order table, create the following functionality:

- **It should always equal “1” if an order is placed through the checkout**
- **It should always equal “0” if it was placed in the Admin**

Solution

1. Create events.xml for your observer in the adminhtml area and tie it to the sales_order_place_after event:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright Â© Magento, Inc. 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="sales_order_place_after">
        <observer name="require_verification_adminhtml"
instance="Unit2\RequireVerification\Observer\Adminhtml\Verification"/>
    </event>
</config>
```

2. Create the observer class for this observer:

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

use Magento\Framework\Event\Observer;
use Magento\Framework\Event\ObserverInterface;
use Psr\Log\LoggerInterface;
```

```
use Magento\Framework\App\State;

class Verification implements ObserverInterface
{
    private $logger;

    private $state;

    /**
     * __construct
     *
     * @return void
     */
    public function __construct(
        LoggerInterface $logger,
        State $state
        )
    {
        $this->logger = $logger;
        $this->state = $state;
    }

    /**
     * execute
     *
     * @param  mixed $observer
     *
     * @return void
     */
    public function execute(Observer $observer)
    {
        $order = $observer->getEvent()->getOrder();
        $order->setRequireVerification(0)->save();
```

```
        $this->logger->info('order saving...' . PHP_EOL, [$this->state->getAreaCode(),
$observer->getEvent()->getOrder()->getData()]);
    }
}
```

3. Create events.xml for your observer in the global area and tie it to the sales_order_place_after event:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright Â© Magento, Inc. 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="sales_order_place_after">
        <observer name="require_verification"
instance="Unit2\RequireVerification\Observer\Frontend\Verification"/>
    </event>
</config>
```

4. Create the observer class for this observer:

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

use Magento\Framework\Event\Observer;
use Magento\Framework\Event\ObserverInterface;
use Psr\Log\LoggerInterface;
use Magento\Framework\App\State;
```

```
class Verification implements ObserverInterface
{
    private $logger;

    private $state;

    /**
     * __construct
     *
     * @return void
     */
    public function __construct(
        LoggerInterface $logger,
        State $state
        )
    {
        $this->logger = $logger;
        $this->state = $state;
    }

    /**
     * execute
     *
     * @param  mixed $observer
     *
     * @return void
     */
    public function execute(Observer $observer)
    {
        $order = $observer->getEvent()->getOrder();
        $order->setRequireVerification(1)->save();
```



```
        $this->logger->info('order saving...' . PHP_EOL, [$this->state->getAreaCode(),
$observer->getEvent()->getOrder()->getData()]);
    }
}
```

2.1.8.1

Create a table, grid, grid filters.

Create a new table, computer_games with the following fields:

- **game_id**
- **name (varchar)**
- **type (rpg, rts, mmo, simulator, shooter)**
- **trial_period (period in days)**
- **release_date**

Compose for the game table its model resource model and collection.

For this table, create a grid with columns that corresponds to fields.

Make release_date column optional (not visible by default).

Create filters that corresponds to fields (text for name, dropdown for type, dropdown for trial_perod, date for release_date).

Solution

1. Create a file app/code/Unit1/ComputerGames/etc/module.xml:

```
<?xml version="1.0"?>
<!--
/**
* Copyright Â© 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="Unit2_ComputerGames"/>
</config>
```

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

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

```
 */
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'Unit2_ComputerGames',
    __DIR__
);
```

3. Create the a db_schema.xml:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright Â© Magento, Inc. 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/schema.
xsd">
    <table name="computer_games" engine="innodb" comment="Computer games table"
resource="default">
        <column xsi:type="int" name="game_id" padding="10" unsigned="true"
nullable="false" identity="true" comment="Primary"/>
        <column xsi:type="text" name="name" nullable="false" comment="Name"/>
        <column xsi:type="text" name="type" nullable="false" comment="Game type: RPG,
Simulator, Shooter, RTS, MMO"/>
        <column xsi:type="int" name="trial_period" padding="3" unsigned="true"
nullable="false" comment="Trial period"/>
        <column xsi:type="date" name="release_date" nullable="false" comment="Release
date"/>
        <constraint xsi:type="primary" referenceId="PRIMARY">
            <column name="game_id"/>
        </constraint>
    </table>
</schema>
```

4. Model, Resource Model, and Collection classes:

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

use Unit2\ComputerGames\Model\ResourceModel\Game as GameResourceModel;

/**
 * Class Game
 * @package Unit2\ComputerGames\Model
 */
class Game extends \Magento\Framework\Model\AbstractExtensibleModel
{
    /**
     * Initialize resource model
     *
     * @return void
     */
    protected function _construct()
    {
        $this->_init(GameResourceModel::class);
    }

    /**
     * @return array
     */
    public function getCustomAttributesCodes()
    {
        return array('game_id', 'name', 'type', 'trial_period', 'release_date', 'image');
```

```
    }
}
```

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

/**
 * Class Game
 * @package Unit2\ComputerGames\Model\ResourceModel
 */
class Game extends \Magento\Framework\Model\ResourceModel\Db\AbstractDb
{
    /**
     * Constructor
     */
    protected function _construct()
    {
        $this->_init('computer_games', 'game_id');
    }
}
```

```
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\ComputerGames\Model\ResourceModel\Game;
use Unit2\ComputerGames\Model\Game as Model;
use Unit2\ComputerGames\Model\ResourceModel\Game as ResourceModel;
```

```
use Magento\Framework\Api\Search\SearchResultInterface;


/**
 * Class Collection
 * @package Unit2\ComputerGames\Model\ResourceModel\Game
 */
class Collection extends
\Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection
    implements SearchResultInterface
{
    /**
     * Constructor
     */
    protected function _construct()
    {
        $this->_init(Model::class, ResourceModel::class);
    }


    /**
     * @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 $this->searchCriteria;
    }

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

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

```
    }

    /**
     * Set total count.
     *
     * @param int $totalCount
     * @return $this
     */
    public function setTotalCount($totalCount)
    {
        $this->setSize($totalCount);
        return $this;
    }

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

}
```

5. Create the grid for this table. First, you should make a new `adminhtml` page. So, we start by adding the `adminhtml` route:

```
<?xml version="1.0"?>
<!--
/**
* Copyright Â© 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="games" frontName="games">
            <module name="Unit2_ComputerGames" before="Magento_Backend"/>
        </route>
    </router>
</config>
```

Then we need a an adminhtml menu item that leads to a grid page

menu.xml:

```
<?xml version="1.0"?>
<!--
/**
* Copyright Â© 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="Unit2_ComputerGames::games_list" title="PC Games"
module="Unit2_ComputerGames"
             parent="Magento_Catalog::catalog" resource="Unit2_ComputerGames::grid"
             sortOrder="0" action="games/game/index"/>
    </menu>
</config>
```

Now add a grid index backend action class:

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

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

/**
 * Class Index
 * @package Unit2\ComputerGames\Controller\Adminhtml\Game
 */
class Index extends Action
{
    /**
     * ACL access restriction
     */
    const ADMIN_RESOURCE = 'Unit2_ComputerGames::grid';

    /**
     * @return \Magento\Framework\View\Result\Page
     */
    public function execute()
    {
        $backendPage = $this->resultFactory->create(ResultFactory::TYPE_PAGE);

        $backendPage->setActiveMenu('Unit2_ComputerGames::grid');
        $backendPage->addBreadcrumb(__('Dashboard'),__('Games'));
        $backendPage->getConfig()->getTitle()->prepend(__('Games'));
```

```
        return $backendPage;
    }
}
```

Add the grid on just created page.

First, create a layout file for this page and add a grid UI component to the content container.

games_game_index.xml

```
<?xml version="1.0"?>
<!--
/**
 * Copyright Â© 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">
    <update handle="styles"/>
    <body>
        <referenceContainer name="content">
            <uiComponent name="computer_games_listing"/>
        </referenceContainer>
    </body>
</page>
```

specify the configuration of the `computer_games_listing` UiComponent, and it's not the easiest part of the task:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
/**
 * Copyright Â© Magento, Inc. All rights reserved.
```

```
 * See COPYING.txt for license details.
 */
-->
<listing xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configuration.xsd
">
    <argument name="data" xsi:type="array">
        <item name="js_config" xsi:type="array">
            <item name="provider"
xsi:type="string">computer_games_listing.computer_games_listing_data_source</item>
        </item>
    </argument>
    <settings>
        <buttons>
            <button name="add">
                <url path="*/game/edit"/>
                <class>primary</class>
                <label translate="true">Add New Game</label>
                <aclResource>Magento_Sales::create</aclResource>
            </button>
        </buttons>
        <spinner>computer_games_columns</spinner>
        <deps>
            <dep>computer_games_listing.computer_games_listing_data_source</dep>
        </deps>
    </settings>
    <dataSource name="computer_games_listing_data_source"
component="Magento_Ui/js/grid/provider">
        <settings>
            <updateUrl path="mui/index/render"/>
        </settings>
        <aclResource>Unit2_ComputerGames::grid</aclResource>
        <dataProvider class="ComputerGamesGridDataProvider"
name="computer_games_listing_data_source">
```

```
            <settings>
                <requestFieldName>id</requestFieldName>
                <primaryFieldName>main_table.game_id</primaryFieldName>
            </settings>
        </dataProvider>
    </dataSource>
    <listingToolbar name="listing_top">
        <settings>
            <sticky>true</sticky>
        </settings>
        <bookmark name="bookmarks"/>
        <columnsControls name="columns_controls"/>
        <exportButton name="export_button"/>
        <filterSearch name="fulltext"/>
        <filters name="listing_filters" />
        <paging name="listing_paging"/>
    </listingToolbar>
    <columns name="computer_games_columns">
        <settings>
            <childDefaults>
                <param name="fieldAction" xsi:type="array">
                    <item name="provider"
xsi:type="string">computer_games_listing.computer_games_listing.computer_games_column
s.actions</item>
                    <item name="target" xsi:type="string">applyAction</item>
                    <item name="params" xsi:type="array">
                        <item name="0" xsi:type="string">view</item>
                        <item name="1" xsi:type="string">${ $.$data.rowIndex }</item>
                    </item>
                </param>
            </childDefaults>
        </settings>
        <column name="game_id" sortOrder="10">
            <settings>
```

```
                <filter>textRange</filter>
                <label translate="true">ID</label>
                <sorting>asc</sorting>
            </settings>
        </column>
        <column name="name">
            <settings>
                <filter>text</filter>
                <label translate="true">Name</label>
            </settings>
        </column>
        <column name="type" component="Magento_Ui/js/grid/columns/select">
            <settings>
                <options class="Unit2\ComputerGames\Ui\Component\Options\Types"/>
                <filter>select</filter>
                <dataType>select</dataType>
                <label translate="true">Type</label>
            </settings>
        </column>
        <column name="trial_period">
            <settings>
                <options
class="Unit2\ComputerGames\Ui\Component\Options\TrialPeriods"/>
                <filter>select</filter>
                <dataType>select</dataType>
                <label translate="true">Trial Period</label>
            </settings>
        </column>
        <column name="release_date" class="Magento\Ui\Component\Listing\Columns\Date"
component="Magento_Ui/js/grid/columns/date">
            <settings>
                <filter>dateRange</filter>
                <dataType>date</dataType>
                <label translate="true">Release Date</label>
```

```
                <sorting>desc</sorting>
                <visible>false</visible>
            </settings>
        </column>
        <column name="image"
class="Unit2\ComputerGames\Ui\Component\Listing\Column\Image"
component="Magento_Ui/js/grid/columns/thumbnail" sortOrder="0" displayArea="general-
area">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="source" xsi:type="string">games</item>
                </item>
            </argument>
            <settings>
                <label translate="true">Image</label>
                <hasPreview>1</hasPreview>
            </settings>
        </column>
        <actionsColumn name="actions"
class="Unit2\ComputerGames\Ui\Component\Listing\Column\GameActions" sortOrder="200">
            <settings>
                <indexField>entity_id</indexField>
            </settings>
        </actionsColumn>
    </columns>
</listing>
```

Specify types for a type column and actions for the grid:

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

```
namespace Unit2\ComputerGames\Ui\Component\Options;

class Types implements \Magento\Framework\Data\OptionSourceInterface
{
    const TYPE_OPTIONS = [
            ['label' => 'RPG',       'value' => 'RPG'],
            ['label' => 'RTS',       'value' => 'RTS'],
            ['label' => 'MMO',       'value' => 'MMO'],
            ['label' => 'Simulator', 'value' => 'Simulator'],
            ['label' => 'Shooter',   'value' => 'Shooter']
        ];

    /**
     * Get options
     *
     * @return array
     */
    public function toOptionArray()
    {

        $this->options = self::TYPE_OPTIONS;
        return $this->options;
    }
}
```

Specify mass action for an actionsColumn:

```
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\ComputerGames\Ui\Component\Listing\Column;
```

```
use Magento\Framework\View\Element\UiComponent\ContextInterface;
use Magento\Framework\View\Element\UiComponentFactory;
use Magento\Ui\Component\Listing\Columns\Column;
use Magento\Cms\Block\Adminhtml\Page\Grid\Renderer\Action\UrlBuilder;
use Magento\Framework\UrlInterface;

class GameActions extends Column
{
    /**
     * Url paths
     */
    const GAME_URL_PATH_EDIT   = 'games/game/edit';
    const GAME_URL_PATH_DELETE = 'games/game/delete';

    /**
     * @var UrlBuilder
     */
    protected $actionUrlBuilder;

    /**
     * @var UrlInterface
     */
    protected $urlBuilder;

    /**
     * @var string
     */
    private $editUrl;

    /**
     * @param ContextInterface $context
     * @param UiComponentFactory $uiComponentFactory
     * @param UrlBuilder $actionUrlBuilder
```

```
     * @param UrlInterface $urlBuilder
     * @param array $components
     * @param array $data
     * @param string $editUrl
     */
    public function __construct(
        ContextInterface $context,
        UiComponentFactory $uiComponentFactory,
        UrlBuilder $actionUrlBuilder,
        UrlInterface $urlBuilder,
        array $components = [],
        array $data = [],
        $editUrl = self::GAME_URL_PATH_EDIT
    ) {
        $this->urlBuilder = $urlBuilder;
        $this->actionUrlBuilder = $actionUrlBuilder;
        $this->editUrl = $editUrl;
        parent::__construct($context, $uiComponentFactory, $components, $data);
    }

    /**
     * Prepare Data Source
     *
     * @param array $dataSource
     * @return array
     */
    public function prepareDataSource(array $dataSource)
    {
        if (isset($dataSource['data']['items'])) {
            foreach ($dataSource['data']['items'] as & $item) {
                $name = $this->getData('name');
                if (isset($item['game_id'])) {
                    $item[$name]['edit'] = [
```

```
                        'href' => $this->urlBuilder->getUrl($this->editUrl,
['game_id' => $item['game_id']]),
                        'label' => __('Edit')
                    ];
                    $item[$name]['delete'] = [
                        'href' => $this->urlBuilder-
>getUrl(self::GAME_URL_PATH_DELETE, ['game_id' => $item['game_id']]),
                        'label' => __('Delete'),
                        'confirm' => [
                            'title' => __('Delete ${ $.$data.name }'),
                            'message' => __('Are you sure you wan\'t to delete a ${
$.$data.name } record?')
                        ]
                    ];
                }
            }
        }

        return $dataSource;
    }
}
```

create a di.xml file specifying data provider virtual type in the grid config XML:

```
<virtualType name="ComputerGamesGridDataProvider"

type="Magento\Framework\View\Element\UiComponent\DataProvider\DataProvider">
        <arguments>
            <argument name="collection" xsi:type="object"
shared="false">Unit2\ComputerGames\Model\ResourceModel\Game\Collection</argument>
        </arguments>
    </virtualType>


    <type
name="Magento\Framework\View\Element\UiComponent\DataProvider\CollectionFactory">
```

```
        <arguments>
            <argument name="collections" xsi:type="array">
                <item name="computer_games_listing_data_source"
xsi:type="string">Unit2\ComputerGames\Model\ResourceModel\Game\Collection</item>
            </argument>
        </arguments>
    </type>
```

Make release_date column optional (not visible by default):

```
<column name="release_date" class="Magento\Ui\Component\Listing\Columns\Date"
component="Magento_Ui/js/grid/columns/date">
            <settings>
                <filter>dateRange</filter>
                <dataType>date</dataType>
                <label translate="true">Release Date</label>
                <sorting>desc</sorting>
                <visible>false</visible>
            </settings>
        </column>
```

Item filter creates a filter that corresponds to each field (text for name, dropdown for type, dropdown for trial_perod, date for release_date).

```
<item name="filter" xsi:type="string">filter type here</item>
```

2.1.8.2

Insert a JavaScript module to a Computer Games module's grid page from the previous exercise. Get access to the visible grid's data in this JavaScript module. (Optionally) add a new column to this grid and pass data from the property to the column.

Solution

1. Create new JavaScript component:
   `Unit2\AccessData\view\adminhtml\web\js\access_data.js`

```
/**
 * Copyright Â© Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
define(["Magento_Ui/js/grid/columns/column", "jquery"], (Component, $) => {
    "use strict";

    return Component.extend({
        defaults: {
            accessData: 'test',
            imports: {
                access_data: '${$.provider}:data'
            }
        },

        initialize: function () {
            this._super();
        },

        access_data: function (access_data) {
            console.log('accessData2 ', access_data);
        },

        getLabel: function () {
            return this.accessData;
        },
    });
});
```

2. specify the configuration of the column for `computer_games_listing` UiComponent:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
/**
```

```
 * Copyright Â© Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<listing xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configuration.xsd
">
    <columns name="computer_games_columns">
        <column name="test" sortOrder="50"
component="Unit2_AccessData/js/access_data">
            <settings>
                <filter>text</filter>
                <label translate="true">Test</label>
            </settings>
        </column>
    </columns>
</listing>
```

2.2 Introduction to Forms

2.2.4.1

Create edit/new form, for the computer_games table and grid created in previous section, add a form.

Use following field types:

- **name (text)**
- **type (dropdown: RPG, RTS, MMO, Simulator, Shooter)**
- **trial_period (integer)**
- **release_date (date)**

Make sure the save/delete/back buttons are available and functioning

Solution

1. Create action class for the form page:

```
<?php
```

```
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\ComputerGames\Controller\Adminhtml\Game;
use Magento\Backend\App\Action;
use Magento\Framework\Registry;
use Magento\Framework\View\Result\PageFactory;

/**
 * Class Edit
 * @package Unit2\ComputerGames\Controller\Adminhtml\Game
 */
class Edit extends Action
{
    /**
     * @var PageFactory
     */
    protected $resultPageFactory;
    /**
     * @var Registry
     */
    protected $coreRegistry;

    /**
     * __construct
     *
     * @return void
     */
    public function __construct(
        PageFactory $resultPageFactory,
        Registry $coreRegistry,
        Action\Context $context
```

```
    )
    {
        $this->resultPageFactory = $resultPageFactory;
        $this->coreRegistry = $coreRegistry;
        parent::__construct($context);
    }

    /**
     * @return \Magento\Framework\View\Result\Page
     */
    public function execute()
    {
        $resultPage = $this->resultPageFactory->create();
        $resultPage->getConfig()->getTitle()->prepend(__('Edit Game'));
        $gameId = $this->getRequest()->getParam('game_id');
        $this->coreRegistry->register('game_id', $gameId);

        return $resultPage;
    }

    /**
     * _isAllowed
     *
     * @return void
     */
    protected function _isAllowed()
    {
        return $this->_authorization->isAllowed('Unit2_ComputerGames::grid');
    }
}
```

2. Add edit form layout:

```
<?xml version="1.0"?>
```

```
<!--
/**
 * Copyright Â© Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" layout="admin-2columns-
left"

xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configurati
on.xsd">
    <body>
        <referenceContainer name="content">
            <uiComponent name="computer_games_form"/>
        </referenceContainer>
    </body>
</page>
```

3. Add the UI component (form) `config xml` file and fields:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
/**
 * Copyright Â© Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<form xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configuration.xsd
">
    <argument name="data" xsi:type="array">
        <item name="js_config" xsi:type="array">
            <item name="provider"
xsi:type="string">computer_games_form.game_form_data_source</item>
        </item>
```

```
        <item name="label" xsi:type="string" translate="true">Computer Game
Information</item>
        <item name="template" xsi:type="string">templates/form/collapsible</item>
    </argument>
    <settings>
        <buttons>
            <button name="save"
class="Unit2\ComputerGames\Block\Adminhtml\Edit\SaveButton"/>
            <button name="delete"
class="Unit2\ComputerGames\Block\Adminhtml\Edit\DeleteButton"/>
            <button name="back"
class="Unit2\ComputerGames\Block\Adminhtml\Edit\BackButton"/>
           <button name="save_and_continue"
class="Unit2\ComputerGames\Block\Adminhtml\Edit\SaveAndContinueButton"/>
        </buttons>
        <namespace>computer_games_form</namespace>
        <dataScope>data</dataScope>
        <deps>
            <dep>computer_games_form.game_form_data_source</dep>
        </deps>
    </settings>
    <dataSource name="game_form_data_source">
        <argument name="data" xsi:type="array">
            <item name="js_config" xsi:type="array">
                <item name="component"
xsi:type="string">Magento_Ui/js/form/provider</item>
            </item>
        </argument>
        <settings>
            <submitUrl path="games/game/save"/>
        </settings>
        <dataProvider class="Unit2\ComputerGames\Ui\Component\Form\DataProvider"
name="game_form_data_source">
            <settings>
                <requestFieldName>game_id</requestFieldName>
```

```
            <primaryFieldName>game_id</primaryFieldName>
        </settings>
    </dataProvider>
</dataSource>
<fieldset name="general">
    <settings>
        <componentType>fieldset</componentType>
        <label translate="true">Information</label>
    </settings>
    <field name="game_id" formElement="input">
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="source" xsi:type="string">computer_games</item>
            </item>
        </argument>
        <settings>
            <dataType>text</dataType>
            <visible>false</visible>
            <dataScope>game_id</dataScope>
        </settings>
    </field>
    <field name="name" sortOrder="20" formElement="input">
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="source" xsi:type="string">computer_games</item>
            </item>
        </argument>
        <settings>
            <validation>
                <rule name="required-entry" xsi:type="boolean">true</rule>
            </validation>
            <dataType>text</dataType>
            <label translate="true">Name</label>
```

```
                <dataScope>name</dataScope>
            </settings>
        </field>
        <field name="type" formElement="select">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="source" xsi:type="string">computer_games</item>
                </item>
            </argument>
            <settings>
                <dataType>number</dataType>
                <label translate="true">Type</label>
                <dataScope>type</dataScope>
            </settings>
            <formElements>
                <select>
                    <settings>
                        <options
class="Unit2\ComputerGames\Ui\Component\Options\Types"/>
                    </settings>
                </select>
            </formElements>
        </field>
        <field name="trial_period" formElement="select">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="source" xsi:type="string">computer_games</item>
                </item>
            </argument>
            <settings>
                <dataType>number</dataType>
                <label translate="true">Trial Period</label>
                <dataScope>trial_period</dataScope>
```

```
            </settings>
            <formElements>
                <select>
                    <settings>
                        <options
class="Unit2\ComputerGames\Ui\Component\Options\TrialPeriods"/>
                    </settings>
                </select>
            </formElements>
        </field>
        <field name="release_date" formElement="date">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="source" xsi:type="string">computer_games</item>
                </item>
            </argument>
            <settings>
                <validation>
                    <rule name="required-entry" xsi:type="boolean">true</rule>
                    <rule name="validate-date" xsi:type="boolean">true</rule>
                </validation>
                <dataType>text</dataType>
                <label translate="true">Release Date</label>
                <dataScope>release_date</dataScope>
            </settings>
        </field>
    </fieldset>
</form>
```

4. Make sure all form buttons work. Create each button block, write action handler classes for them, then create a validate action class to make sure the submit form works.

2.2.4.2

Focusing on the previous task, create an image uploader for the edit/new form, for the computer_games table created in previous section.

Make image field as the required form field.

Make sure the imageUploader is available and functioning.

2.2.4.3

Make image field available on the grid page.

Solution

1. Add new filed to the Ui Form with `formElement imageUploader` component:

```
<field name="image" formElement="imageUploader">
            <settings>
                <validation>
                    <rule name="required-entry" xsi:type="boolean">true</rule>
                </validation>
                <notice translate="true">Allowed file types: jpeg, gif, png.</notice>
                <label translate="true">Image</label>
                <componentType>imageUploader</componentType>
            </settings>
            <formElements>
                <imageUploader>
                    <settings>
                        <maxFileSize>2097152</maxFileSize>
                        <uploaderConfig>
                            <param xsi:type="string"
name="url">games/game_fileUploader/save</param>
                        </uploaderConfig>
                    </settings>
                </imageUploader>
            </formElements>
        </field>
```

2. Add configuration to the a `di.xml` file:

```
<virtualType name="gamesImageUploader" type="Magento\Catalog\Model\ImageUploader">
        <arguments>
            <argument name="baseTmpPath" xsi:type="string">tmp/upload/games</argument>
            <argument name="basePath" xsi:type="string">games/upload</argument>
            <argument name="allowedExtensions" xsi:type="array">
                <item name="jpg" xsi:type="string">jpg</item>
                <item name="jpeg" xsi:type="string">jpeg</item>
                <item name="gif" xsi:type="string">gif</item>
                <item name="png" xsi:type="string">png</item>
            </argument>
        </arguments>
    </virtualType>

    <type name="Unit2\ComputerGames\Controller\Adminhtml\Game\FileUploader\Save">
        <arguments>
            <argument name="imageUploader"
xsi:type="object">gamesImageUploader</argument>
        </arguments>
    </type>
    <type name="Unit2\ComputerGames\Controller\Adminhtml\Game\Save">
        <arguments>
            <argument name="imageUploader"
xsi:type="object">gamesImageUploader</argument>
        </arguments>
    </type>
<type name="Unit2\ComputerGames\Ui\Component\Listing\Column\Image">
        <arguments>
            <argument name="imageUploader"
xsi:type="object">gamesImageUploader</argument>
        </arguments>
    </type>
    <type name="Unit2\ComputerGames\Ui\Component\Form\ImageModifier">
        <arguments>
```

```
            <argument name="imageUploader"
xsi:type="object">gamesImageUploader</argument>
        </arguments>
    </type>
<virtualType name="GamesImageUploaderUiDataProviderImageFormModifierPool"
type="Magento\Ui\DataProvider\Modifier\Pool">
        <arguments>
            <argument name="modifiers" xsi:type="array">
                <item name="image_data" xsi:type="array">
                    <item name="class"
xsi:type="string">Unit2\ComputerGames\Ui\Component\Form\ImageModifier</item>
                    <item name="sortOrder" xsi:type="number">10</item>
                </item>
            </argument>
        </arguments>
    </virtualType>
    <type name="Unit2\ComputerGames\Ui\Component\Form\DataProvider">
        <arguments>
            <argument name="pool"
xsi:type="object">GamesImageUploaderUiDataProviderImageFormModifierPool</argument>
        </arguments>
    </type>
```

3. Implement a `ModifierInterface`:

```
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\ComputerGames\Ui\Component\Form;

use Magento\Ui\DataProvider\Modifier\ModifierInterface;
use \Magento\Store\Model\StoreManagerInterface;
use \Magento\Catalog\Model\ImageUploader;
```

```
class ImageModifier implements ModifierInterface
{
    protected $storeManager;

    protected $imageUploader;

    public function __construct(
        StoreManagerInterface $storeManager,
        ImageUploader $imageUploader
    ) {
        $this->storeManager = $storeManager;
        $this->imageUploader = $imageUploader;
    }
    /**
     * @param array $meta
     * @return array
     */
    public function modifyMeta(array $meta)
    {
        return $meta;
    }

    /**
     * modifyData
     *
     * @param  mixed $data
     *
     * @return void
     */
    public function modifyData(array $data)
    {
        foreach ($data as $image) {
```



```
            $imageData = $image->getData();
            if (isset($imageData['image'])) {
                $arrayImageData = [];
                $arrayImageData[0]['name'] = 'Image';
                $arrayImageData[0]['url'] = $this->storeManager->getStore()->getBaseUrl(
                    \Magento\Framework\UrlInterface::URL_TYPE_MEDIA
                    ) . $this->imageUploader->getFilePath(
                        $this->imageUploader->getBasePath(),
                        $image->getImage()
                    );
                $imageData['image'] = $arrayImageData;
            }

            $image->setData($imageData);
            $data[$image->getId()] = $imageData;
        }

        return $data;
    }
}
```

4. Add new column to db_shema.xml file:

```
<column xsi:type="text" name="image" nullable="false" comment="Image"/>
```

5. Save the image in the ComputerGames\Controller\Adminhtml\Game\Save.php controller:

```
if (isset($postData['image'])) {
            try {
                $imageName = $this->imageUploader-
>moveFileFromTmp($postData['image'][0]['name']);
                $this->gameFactory->setImage($imageName);
            } catch (\Exception $e) {
                $imageName = ['error' => $e->getMessage(), 'errorcode' => $e->getCode()];
            }
        }
```

2.2.4.4

Create a custom tab on a customer view page. Add a grid from ComputerGames module from previous tasks inside this page.

Solution

1. Create controller and route:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright Â© 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/App/etc/r
outes.xsd">
    <router id="admin">
        <route id="custom" frontName="custom">
            <module name="Unit2_NewCustomerTab" />
        </route>
    </router>
</config>


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

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

class Custom extends Action
```

```
{
    /**
     * ACL access restriction
     */
    const ADMIN_RESOURCE = 'Unit2_NewCustomerTab::grid';



    /**
     * execute
     *
     * @return void
     */
    public function execute()
    {
        $backendPage = $this->resultFactory->create(ResultFactory::TYPE_PAGE);
        return $backendPage;
    }
}
```

2. Create block:

```
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\NewCustomerTab\Block\Adminhtml\Edit\Tab\View;

use Magento\Backend\Block\Template\Context;
use Magento\Backend\Helper\Data;
use Unit2\ComputerGames\Model\GameFactory;
use Magento\Framework\Registry;

class Custom extends \Magento\Backend\Block\Widget\Grid\Extended
```

```
{
    protected $_coreRegistry = null;

    protected $_collectionFactory;

    public function __construct(
        Context $context,
        Data $backendHelper,
        GameFactory $collectionFactory,
        Registry $coreRegistry,
        array $data = []
    ) {

        $this->_coreRegistry = $coreRegistry;
        $this->_collectionFactory = $collectionFactory;
        parent::__construct($context, $backendHelper, $data);
    }

    protected function _construct()
    {
        parent::_construct();
        $this->setDefaultSort('created_at', 'desc');
        $this->setSortable(false);
        $this->setPagerVisibility(false);
        $this->setFilterVisibility(false);
    }
    protected function _prepareGrid()
    {
        $this->setId('computer_games_customer_custom' . $this->getWebsiteId());
        parent::_prepareGrid();
    }

    protected function _prepareCollection()
```

```
{
    $collection = $this->_collectionFactory->create();
    $this->setCollection($collection->getCollection());
    return parent::_prepareCollection();
}

protected function _prepareColumns()
{
    $this->addColumn(
        'game_id',
        [
            'header' => __('ID'),
            'index' => 'game_id',
            'type' => 'number'
        ]
    );

    $this->addColumn(
        'name',
        [
            'header' => __('Name'),
            'index' => 'name'
        ]
    );

    $this->addColumn(
        'type',
        [
            'header' => __('Type'),
            'index' => 'type'
        ]
    );
```

```
        $this->addColumn(
            'trial_period',
            [
                'header' => __('Trial Period'),
                'index' => 'trial_period'
            ]
        );

        $this->addColumn(
            'release_date',
            [
                'header' => __('Telease Date'),
                'index' => 'release_date'
            ]
        );

        return parent::_prepareColumns();
    }
}
```

3. Create a layout and specify a block:

```
<?xml version="1.0"?>
<!--
/**
 * Copyright Â© Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<layout xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configurati
on.xsd">
    <container name="root">
        <block class="Unit2\NewCustomerTab\Block\Adminhtml\Edit\Tab\View\Custom"
name="computer.games.tab.custom"/>
```

```
        </container>
    </layout>
```

4. Create a tab block:

```
<?php
/**
 * Copyright © Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Unit2\NewCustomerTab\Block\Adminhtml\Edit\Tab;

use Magento\Backend\Block\Template;
use Magento\Framework\Registry;
use Magento\Customer\Controller\RegistryConstants;
use Magento\Ui\Component\Layout\Tabs\TabInterface;

class Custom extends Template implements TabInterface
{
    protected $_coreRegistry;

    /**
     * __construct
     *
     * @param  mixed $context
     * @param  mixed $registry
     * @param  mixed $data
     *
     * @return void
     */
    public function __construct(Template\Context $context, Registry $registry, array
$data = [])
    {
        $this->_coreRegistry = $registry;
```

```
        parent::__construct($context, $data);
    }

    /**
     * getCustomerId
     *
     * @return void
     */
    public function getCustomerId()
    {
        return $this->_coreRegistry->registry(RegistryConstants::CURRENT_CUSTOMER_ID);
    }

    /**
     * getTabLabel
     *
     * @return void
     */
    public function getTabLabel()
    {
        return __('Computer Games Tab');
    }

    /**
     * getTabTitle
     *
     * @return void
     */
    public function getTabTitle()
    {
        return __('Computer Games Tab');
    }
```



```
/**
 * canShowTab
 *
 * @return void
 */
public function canShowTab()
{
    if ($this->getCustomerId()) {
        return true;
    }
    return false;
}

/**
 * isHidden
 *
 * @return void
 */
public function isHidden()
{
    if ($this->getCustomerId()) {
        return false;
    }
    return true;
}

/**
 * getTabClass
 *
 * @return void
 */
public function getTabClass()
{
```

```
        return '';
    }

    /**
     * getTabUrl
     *
     * @return void
     */
    public function getTabUrl()
    {
        return $this->getUrl('custom/*/custom', ['_current' => true]);
    }

    /**
     * isAjaxLoaded
     *
     * @return void
     */
    public function isAjaxLoaded()
    {
        return true;
    }
}
```

5. Create a layout and specify a tab block (customer_index_edit.xml):

```
<?xml version="1.0"?>
<!--
/**
 * Copyright Â© Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" layout="admin-2columns-left"
```

```
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configuration.x
sd">
    <body>
            <referenceBlock name="customer_form">
                <block class="Unit2\NewCustomerTab\Block\Adminhtml\Edit\Tab\Custom"
name="customer_edit_tab_custom">
                <action method="setTabLabel">
                    <argument name="label" xsi:type="string">Computer Games
Tab</argument>
                </action>
                </block>
            </referenceBlock>
    </body>
</page>
```

2.2.4.5

Focusing on the previous task create a button for the edit/new form, for the computer_games module which will render the “Hello World” phrase inside the popup.

Solution

1. Add a new field to the UI component (form) `config xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
/**
 * Copyright Â© Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
-->
<form xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configuration.xsd">
    <fieldset name="general">
        <field name="popup" formElement="button"
component="Unit2_CustomButton/js/button">
            <argument name="data" xsi:type="array">
```

```
                <item name="config" xsi:type="array">
                    <item name="title" xsi:type="string">Popup</item>
                </item>
            </argument>
        </field>
    </fieldset>
</form>
```

2. Specify JavaScript component for a new field:

```
/**
 * Copyright Â© Magento. All rights reserved.
 * See COPYING.txt for license details.
 */
define([
    "Magento_Ui/js/form/components/button",
    "Magento_Ui/js/modal/confirm",
    "jquery"
], (Component, modalConfirm, $) => {
    "use strict";

    return Component.extend({

    action: function () {
        modalConfirm({
            content: "Hello World",
            opened: function() {
                $('body').trigger('contentUpdated');
            },
            buttons: [{
                text: 'Cancel',
                class: 'action-primary action-dismiss',

                /**
```

```
                 * Click handler.
                 */
                click: function (event) {
                    this.closeModal(event);
                }
            }]
        });
    }
    });
});
```