Technical Documentation

Document ID:5464 and 5458
Document Version:1.0
Authors:Adam Hussain and Akash
Team-5

1. Overall Description

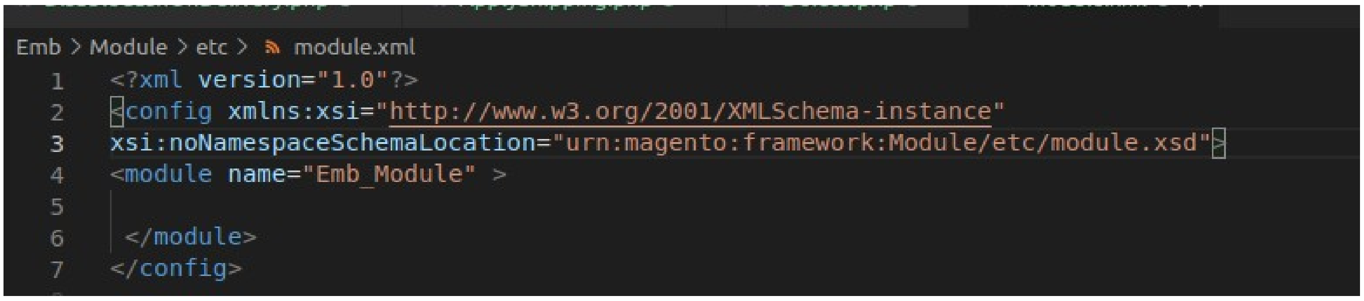
This document contains all the functionalities that we successfully finished.
Here we are creating Grid and From in the magento backend admin panel

2. Tecnical Requirements

2.1 GRID and FORM

Step 1: Creating the Module: In the magento root directory app/code, create the folder to build the extension in a given structure which is defined as in this project the vendor name as 'Emb' and module name as 'Module'.

app/code/Emb/Module/etc/module.xml

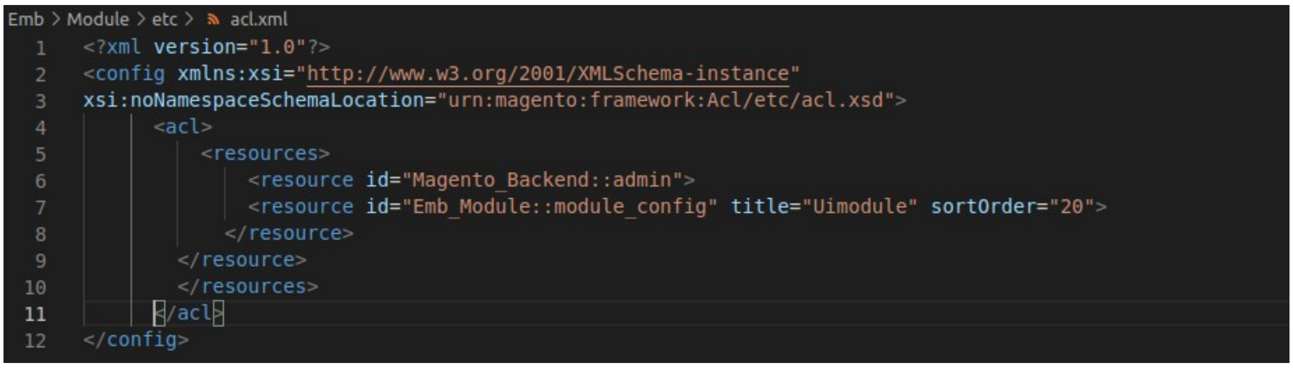


Emb > Module > etc > module.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
<module name="Emb_Module" >

 </module>
</config>
```

Step 2: Create the acl.xml: Magento 2 admin acl use an authentication system and a robust system for create Access Control List Rules (ACL) which allows customer for free shipping to create fine grained roles for each and every user in their system.

app/code/Emb/Module/etc/acl.xml

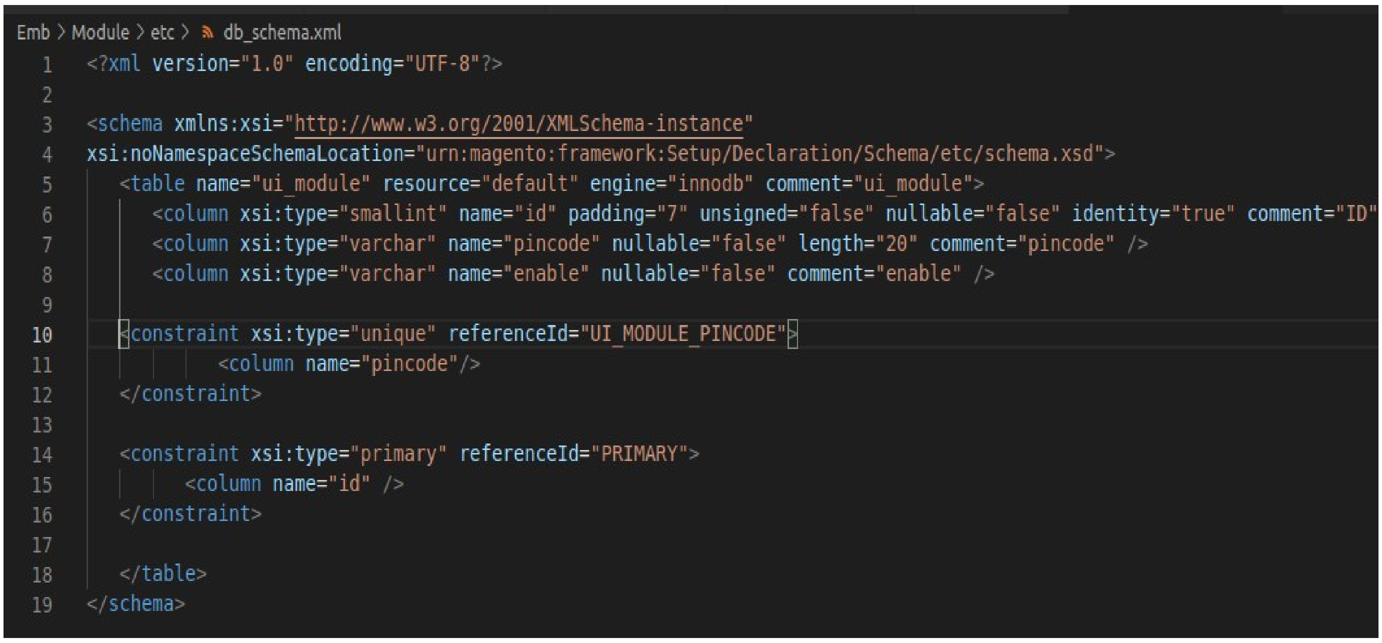


Emb > Module > etc > acl.xml

```xml
<?xml version="1.0"?>
<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="Emb_Module::module_config" title="Uimodule" sortOrder="20">
            </resource>
        </resource>
        </resources>
      </acl>
</config>
```

Step 3: Create the db_schema.xml: This file is used to create the new tables or maintain the tables which avoids the unneccesary version checking for creating new columns or delete columns and also for backward compactibility with php script.

app/code/Emb/Module/etc/db_schema.xml

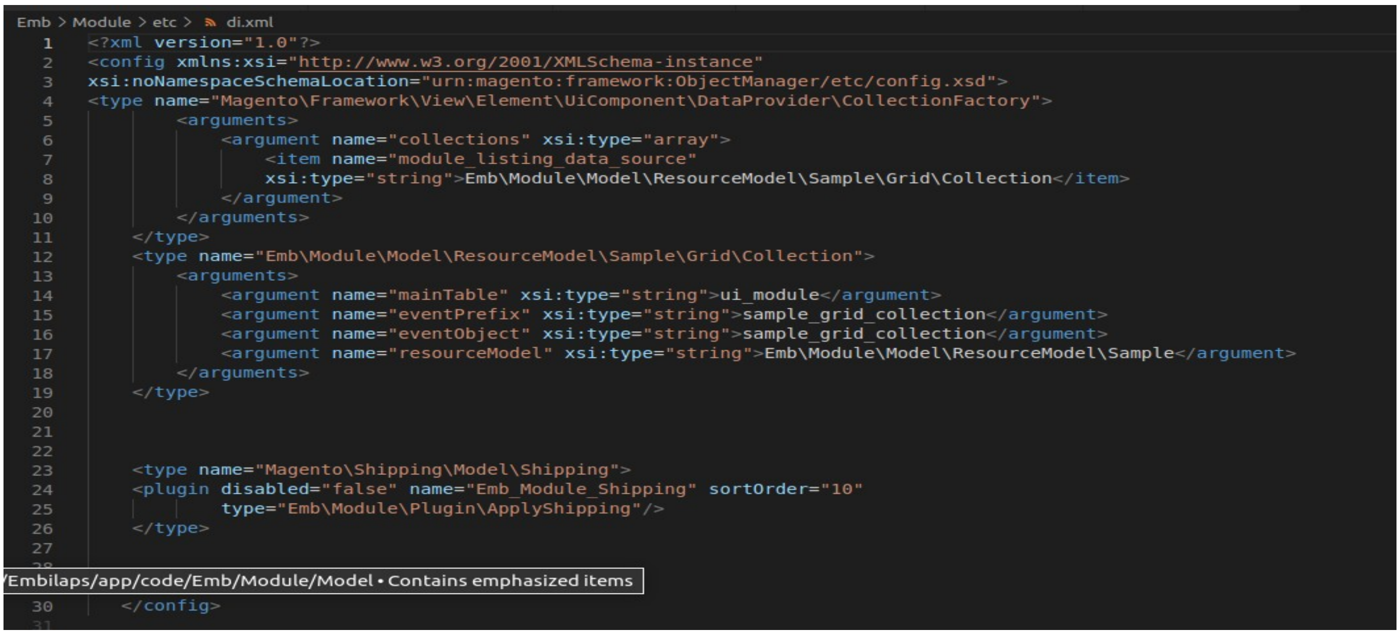


Emb > Module > etc > db_schema.xml

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

<schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Setup/Declaration/Schema/etc/schema.xsd">
   <table name="ui_module" resource="default" engine="innodb" comment="ui_module">
      <column xsi:type="smallint" name="id" padding="7" unsigned="false" nullable="false" identity="true" comment="ID"
      <column xsi:type="varchar" name="pincode" nullable="false" length="20" comment="pincode" />
      <column xsi:type="varchar" name="enable" nullable="false" comment="enable" />

   <constraint xsi:type="unique" referenceId="UI_MODULE_PINCODE">
         <column name="pincode"/>
   </constraint>

   <constraint xsi:type="primary" referenceId="PRIMARY">
         <column name="id" />
   </constraint>

   </table>
</schema>
```

Step 4:Create the di.xml: This file configures which dependencies are injected by object manager and the application reads all the di.xml configuration files which will be merged all together by appending all nodes.

app/code/Emb/Module/etc/di.xml

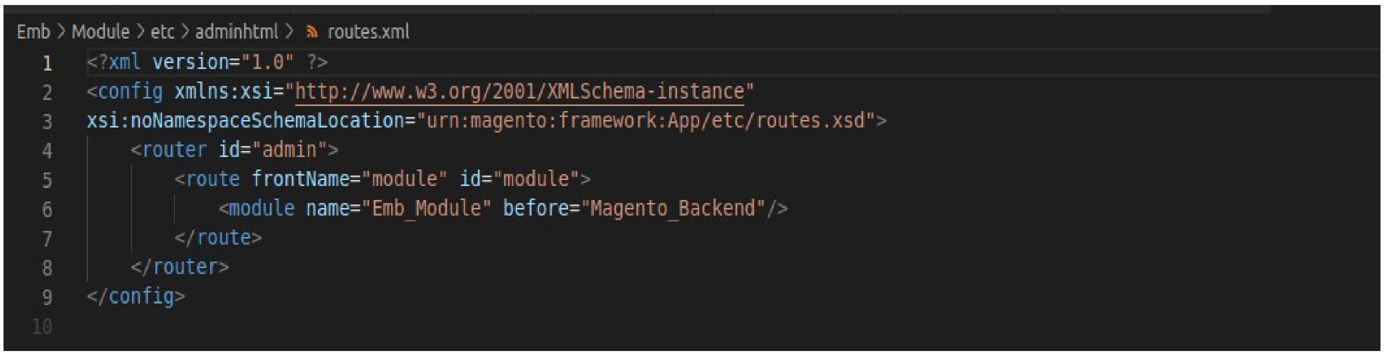


Emb > Module > etc > di.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
<type name="Magento\Framework\View\Element\UiComponent\DataProvider\CollectionFactory">
        <arguments>
            <argument name="collections" xsi:type="array">
                <item name="module_listing_data_source"
                xsi:type="string">Emb\Module\Model\ResourceModel\Sample\Grid\Collection</item>
            </argument>
        </arguments>
    </type>
    <type name="Emb\Module\Model\ResourceModel\Sample\Grid\Collection">
        <arguments>
            <argument name="mainTable" xsi:type="string">ui_module</argument>
            <argument name="eventPrefix" xsi:type="string">sample_grid_collection</argument>
            <argument name="eventObject" xsi:type="string">sample_grid_collection</argument>
            <argument name="resourceModel" xsi:type="string">Emb\Module\Model\ResourceModel\Sample</argument>
        </arguments>
    </type>



    <type name="Magento\Shipping\Model\Shipping">
    <plugin disabled="false" name="Emb_Module_Shipping" sortOrder="10"
            type="Emb\Module\Plugin\ApplyShipping"/>
    </type>

/Embilaps/app/code/Emb/Module/Model • Contains emphasized items
   </config>
```

Step 5: Create the routes.xml in adminhtml: The route will define the name for the module which will be used in the url to find the module and execute the controller action.

app/code/Emb/Module/etc/adminhtml/routes.xml

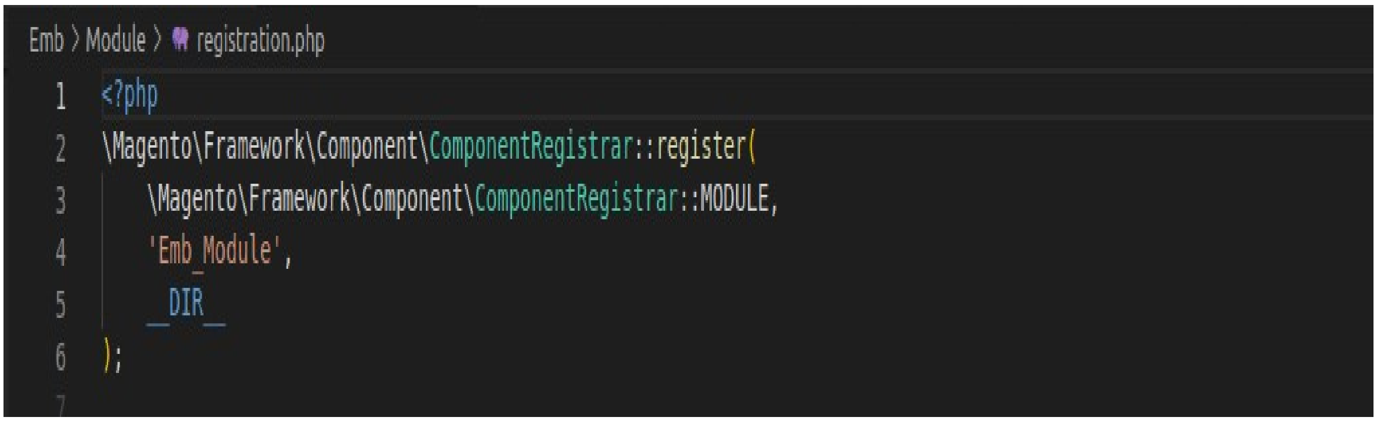


Emb > Module > etc > adminhtml > routes.xml

```xml
<?xml version="1.0" ?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:App/etc/routes.xsd">
    <router id="admin">
        <route frontName="module" id="module">
            <module name="Emb_Module" before="Magento_Backend"/>
        </route>
    </router>
</config>
```

Step 6: Create the registration.php: It is used to notify to magento application about the module in the system. It is also used to provide the location for register the module.

app/code/Emb/Module/registration.php

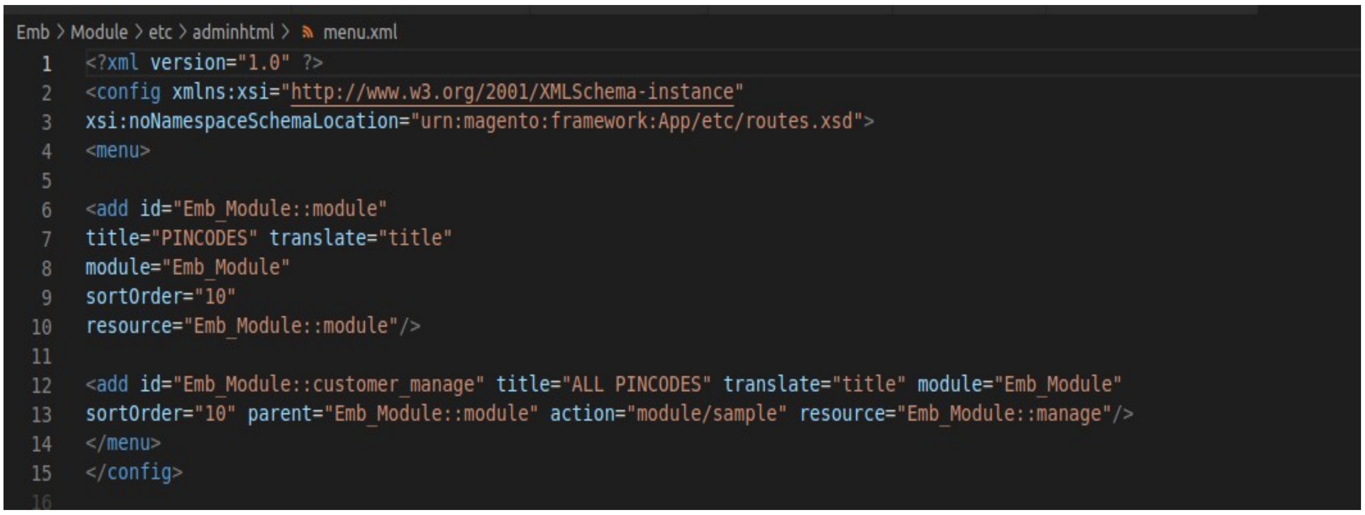


Emb > Module > registration.php

```php
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    \Magento\Framework\Component\ComponentRegistrar::MODULE,
    'Emb_Module',
    __DIR__
);
```

Step 7: Create the menu.xml in adminhtml: It is located in module's etc/adminhtml folder which consists of menu nodes, config and add multiple directives.

app/code/Emb/Module/etc/adminhtml/menu.xml

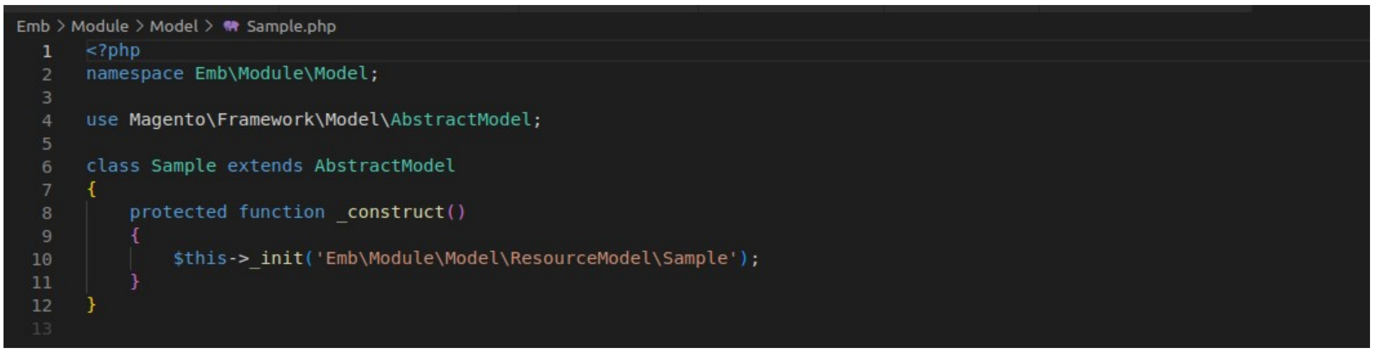


Emb > Module > etc > adminhtml > menu.xml

```xml
<?xml version="1.0" ?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:App/etc/routes.xsd">
<menu>

<add id="Emb_Module::module"
title="PINCODES" translate="title"
module="Emb_Module"
sortOrder="10"
resource="Emb_Module::module"/>

<add id="Emb_Module::customer_manage" title="ALL PINCODES" translate="title" module="Emb_Module"
sortOrder="10" parent="Emb_Module::module" action="module/sample" resource="Emb_Module::manage"/>
</menu>
</config>
```

Step 8: Create the Sample.php in Model: The Sample.php file in the model folder is a PHP class that represents the data model for a specific entity in the system. This file is typically used to define the business logic that relates to that entity, such as saving and retrieving data from the database.

app/code/Emb/Module/Model/Sample.php

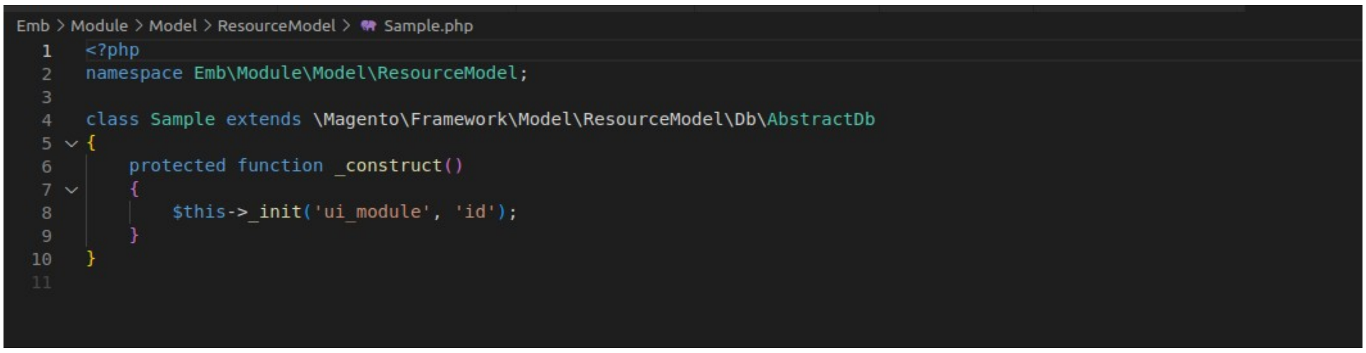


Emb > Module > Model > Sample.php

```php
<?php
namespace Emb\Module\Model;

use Magento\Framework\Model\AbstractModel;

class Sample extends AbstractModel
{
    protected function _construct()
    {
        $this->_init('Emb\Module\Model\ResourceModel\Sample');
    }
}
```

Step 9: Create the Sample.php in Model-ResourceModel folder:

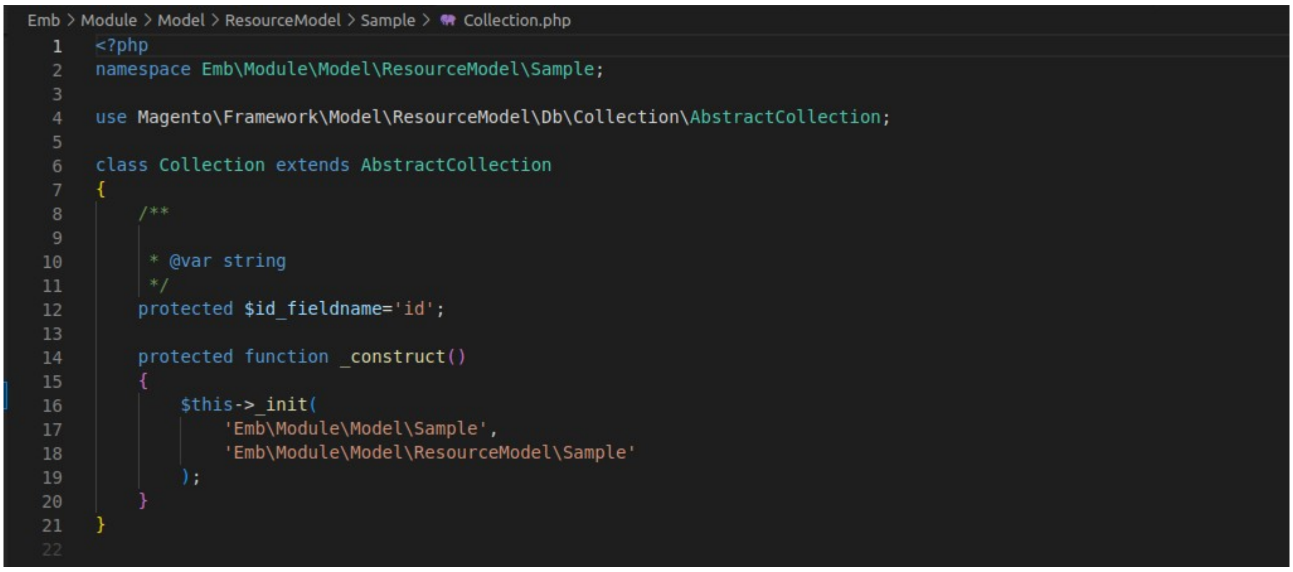
app/code/Emb/Module/Model/ResourceModel/Sample.php


Emb > Module > Model > ResourceModel > Sample.php

```
<?php
namespace Emb\Module\Model\ResourceModel;

class Sample extends \Magento\Framework\Model\ResourceModel\Db\AbstractDb
{
    protected function _construct()
    {
        $this->_init('ui_module', 'id');
    }
}
```

Step 10: Create the Collection.php in Model-ResouceModel-Sample folder: The Collection class in Magento 2 is used to retrieve a set of data from the database. The Collection class is typically used to work with data from custom database tables or to extend core Magento 2 models and work with data from their associated database tables.

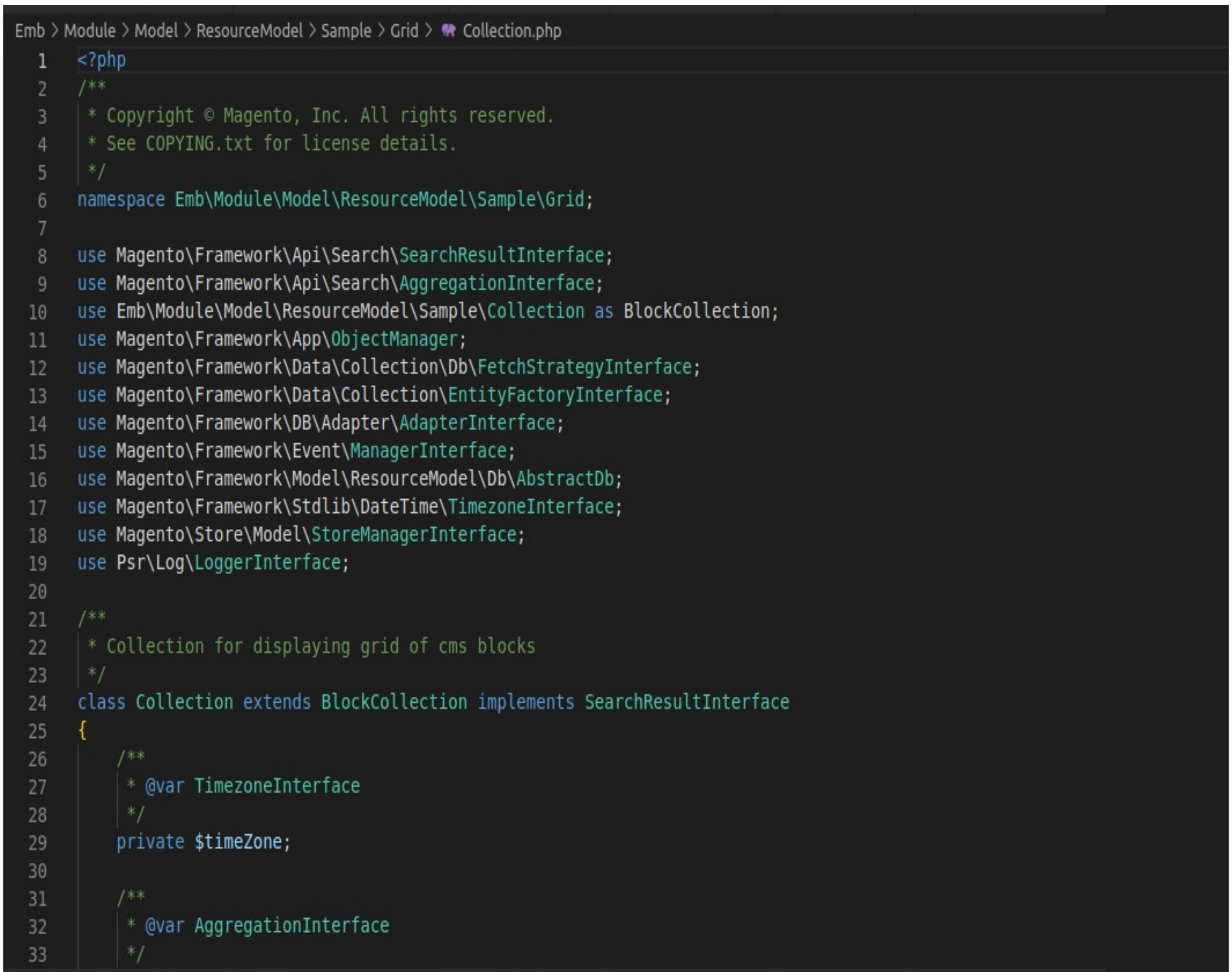
app/code/Emb/Module/Model/ResourceModel/Sample/Collection.php


Emb > Module > Model > ResourceModel > Sample > Collection.php

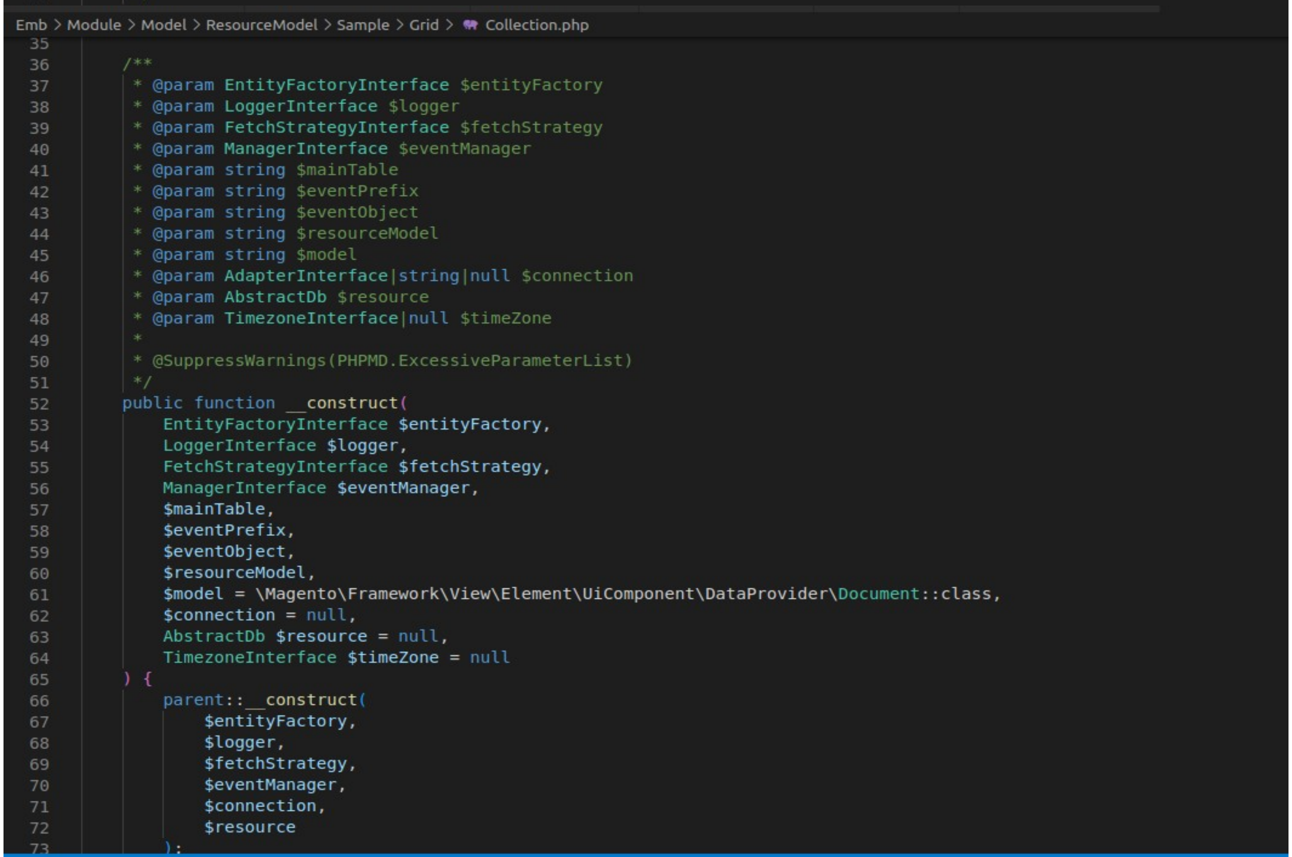
```
<?php
namespace Emb\Module\Model\ResourceModel\Sample;

use Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection;

class Collection extends AbstractCollection
{
    /**

     * @var string
     */
    protected $id_fieldname='id';

    protected function _construct()
    {
        $this->_init(
            'Emb\Module\Model\Sample',
            'Emb\Module\Model\ResourceModel\Sample'
        );
    }
}
```

Step 11: Create the Collection.php in Model-ResouceModel-Sample-Grid folder: In Magento 2, the Collection class for a grid is used to retrieve a set of data that will be displayed in the grid. The Collection class is typically defined in a file called Collection.php, which is located in the Model/ResourceModel directory of a module.

app/code/Emb/Module/Model/ResourceModel/Sample/Grid/Collection.php


Emb > Module > Model > ResourceModel > Sample > Grid > Collection.php

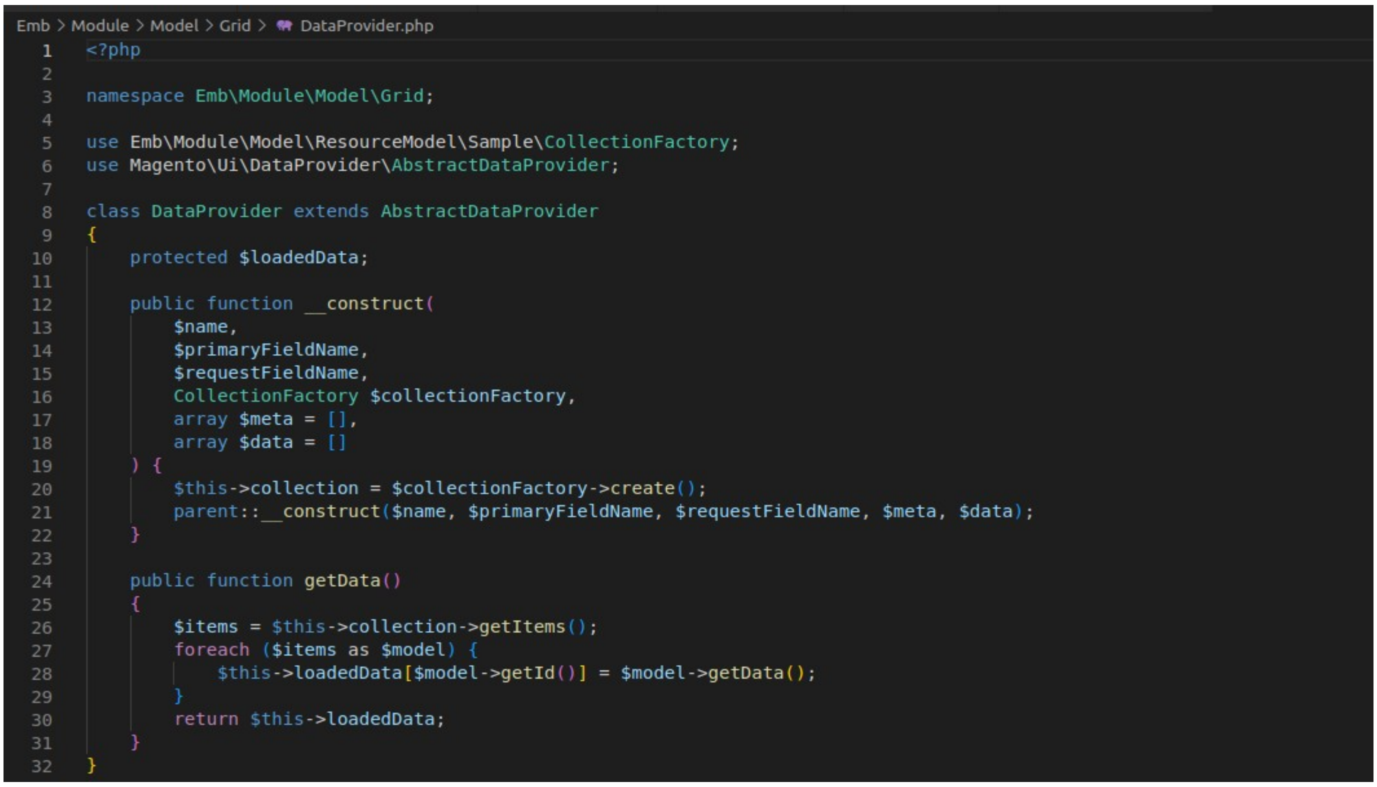
```php
<?php
/**
 * Copyright © Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Emb\Module\Model\ResourceModel\Sample\Grid;

use Magento\Framework\Api\Search\SearchResultInterface;
use Magento\Framework\Api\Search\AggregationInterface;
use Emb\Module\Model\ResourceModel\Sample\Collection as BlockCollection;
use Magento\Framework\App\ObjectManager;
use Magento\Framework\Data\Collection\Db\FetchStrategyInterface;
use Magento\Framework\Data\Collection\EntityFactoryInterface;
use Magento\Framework\DB\Adapter\AdapterInterface;
use Magento\Framework\Event\ManagerInterface;
use Magento\Framework\Model\ResourceModel\Db\AbstractDb;
use Magento\Framework\Stdlib\DateTime\TimezoneInterface;
use Magento\Store\Model\StoreManagerInterface;
use Psr\Log\LoggerInterface;

/**
 * Collection for displaying grid of cms blocks
 */
class Collection extends BlockCollection implements SearchResultInterface
{
    /**
     * @var TimezoneInterface
     */
    private $timeZone;

    /**
     * @var AggregationInterface
     */
```


Emb > Module > Model > ResourceModel > Sample > Grid > Collection.php

```php
    /**
     * @param EntityFactoryInterface $entityFactory
     * @param LoggerInterface $logger
     * @param FetchStrategyInterface $fetchStrategy
     * @param ManagerInterface $eventManager
     * @param string $mainTable
     * @param string $eventPrefix
     * @param string $eventObject
     * @param string $resourceModel
     * @param string $model
     * @param AdapterInterface|string|null $connection
     * @param AbstractDb $resource
     * @param TimezoneInterface|null $timeZone
     *
     * @SuppressWarnings(PHPMD.ExcessiveParameterList)
     */
    public function __construct(
        EntityFactoryInterface $entityFactory,
        LoggerInterface $logger,
        FetchStrategyInterface $fetchStrategy,
        ManagerInterface $eventManager,
        $mainTable,
        $eventPrefix,
        $eventObject,
        $resourceModel,
        $model = \Magento\Framework\View\Element\UiComponent\DataProvider\Document::class,
        $connection = null,
        AbstractDb $resource = null,
        TimezoneInterface $timeZone = null
    ) {
        parent::__construct(
            $entityFactory,
            $logger,
            $fetchStrategy,
            $eventManager,
            $connection,
            $resource
        );
```

Step 12: Create the DataProvider.php: The DataProvider class for a grid is responsible for retrieving data from a collection object and preparing it for display in the grid. The DataProvider class is typically defined in a file called DataProvider.php, which is located in the model directory of a module.

app/code/Emb/Module/Model/Grid/DataProvider.php

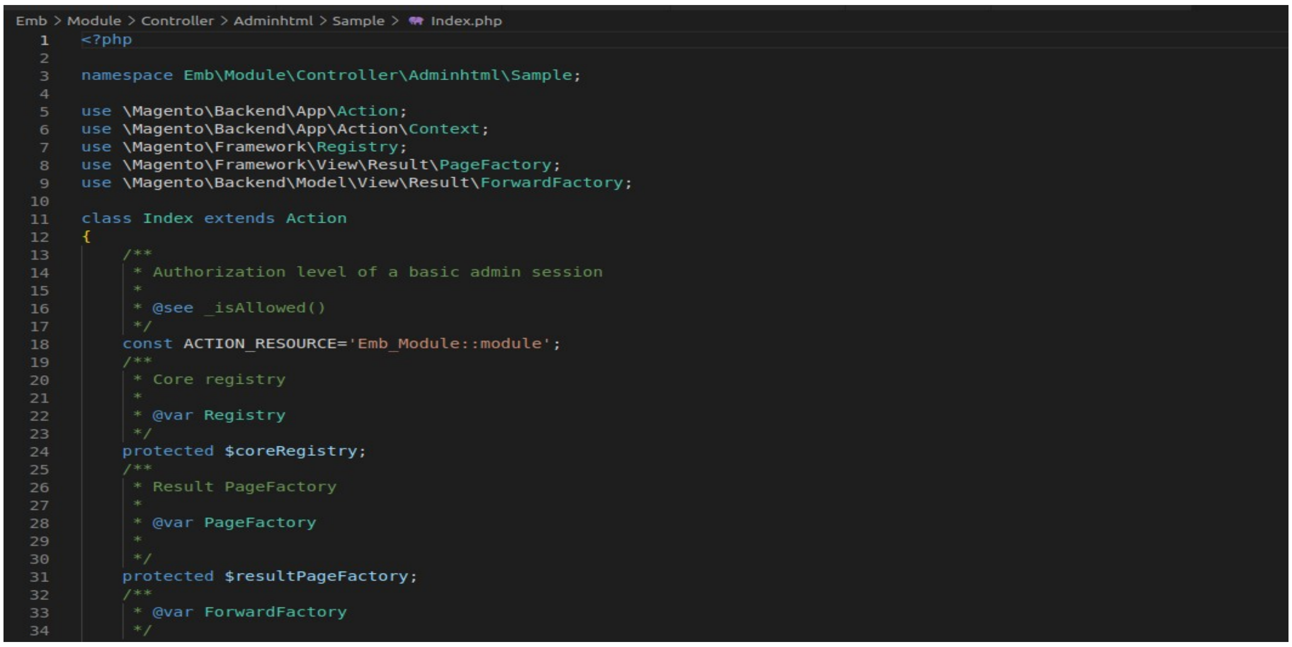


```
Emb > Module > Model > Grid > DataProvider.php
<?php

namespace Emb\Module\Model\Grid;

use Emb\Module\Model\ResourceModel\Sample\CollectionFactory;
use Magento\Ui\DataProvider\AbstractDataProvider;

class DataProvider extends AbstractDataProvider
{
    protected $loadedData;

    public function __construct(
        $name,
        $primaryFieldName,
        $requestFieldName,
        CollectionFactory $collectionFactory,
        array $meta = [],
        array $data = []
    ) {
        $this->collection = $collectionFactory->create();
        parent::__construct($name, $primaryFieldName, $requestFieldName, $meta, $data);
    }

    public function getData()
    {
        $items = $this->collection->getItems();
        foreach ($items as $model) {
            $this->loadedData[$model->getId()] = $model->getData();
        }
        return $this->loadedData;
    }
}
```

Step 13: Create the index.php in the Controller-Adminhtml:

The index.php file located in the adminhtml directory is the main entry point for the Magento admin panel. It is responsible for initializing the admin application and routing incoming requests to the appropriate controller.

app/code/Emb/Module/Controller/Adminhtml/Sample/Index.php

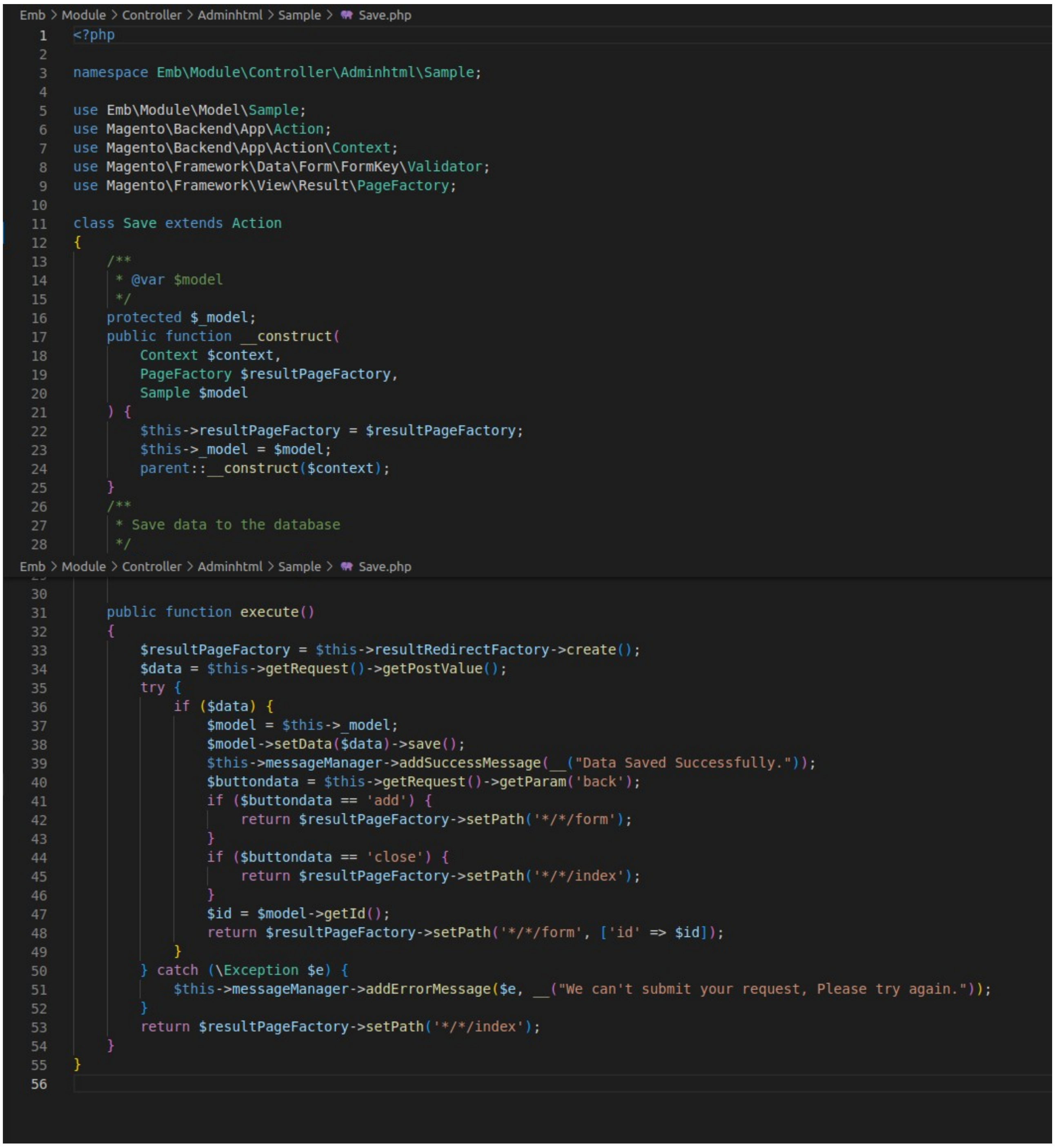


```
Emb > Module > Controller > Adminhtml > Sample > Index.php
<?php

namespace Emb\Module\Controller\Adminhtml\Sample;

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

class Index extends Action
{
    /**
     * Authorization level of a basic admin session
     *
     * @see _isAllowed()
     */
    const ACTION_RESOURCE='Emb_Module::module';
    /**
     * Core registry
     *
     * @var Registry
     */
    protected $coreRegistry;
    /**
     * Result PageFactory
     *
     * @var PageFactory
     *
     */
    protected $resultPageFactory;
    /**
     * @var ForwardFactory
     */
```

Step 14: Create the Save.php file in Controller – Adminhtml:

The save.php file in a custom controller could potentially be responsible for processing a form submission, saving data to the database, or performing **s**ome other action related to data manipulation or persistence.

app/code/Emb/Module/Controller/Adminhtml/Sample/Save.php

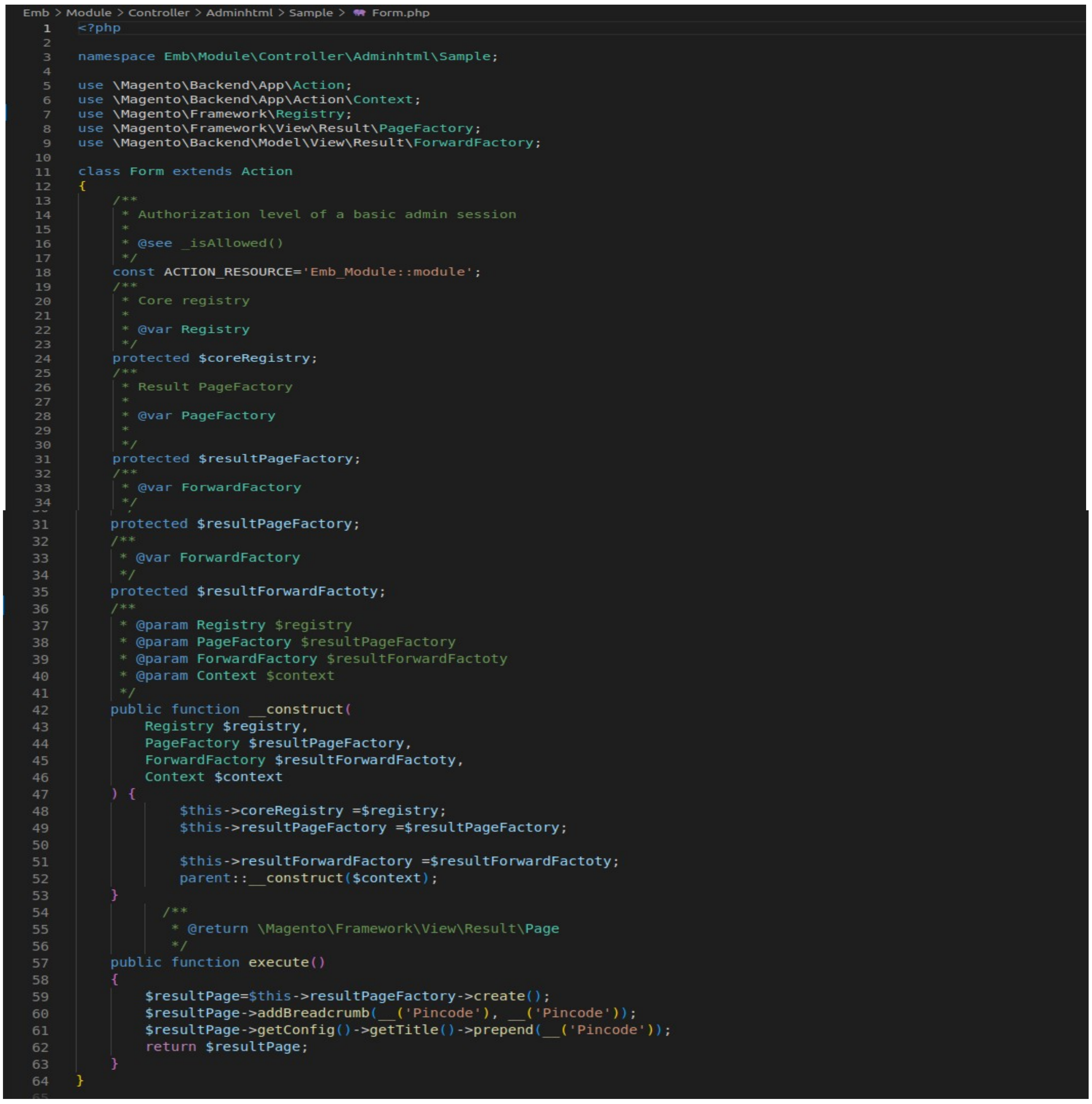


```
Emb > Module > Controller > Adminhtml > Sample > Save.php
<?php

namespace Emb\Module\Controller\Adminhtml\Sample;

use Emb\Module\Model\Sample;
use Magento\Backend\App\Action;
use Magento\Backend\App\Action\Context;
use Magento\Framework\Data\Form\FormKey\Validator;
use Magento\Framework\View\Result\PageFactory;

class Save extends Action
{
    /**
     * @var $model
     */
    protected $_model;
    public function __construct(
        Context $context,
        PageFactory $resultPageFactory,
        Sample $model
    ) {
        $this->resultPageFactory = $resultPageFactory;
        $this->_model = $model;
        parent::__construct($context);
    }
    /**
     * Save data to the database
     */
Emb > Module > Controller > Adminhtml > Sample > Save.php

    public function execute()
    {
        $resultPageFactory = $this->resultRedirectFactory->create();
        $data = $this->getRequest()->getPostValue();
        try {
            if ($data) {
                $model = $this->_model;
                $model->setData($data)->save();
                $this->messageManager->addSuccessMessage(__("Data Saved Successfully."));
                $buttondata = $this->getRequest()->getParam('back');
                if ($buttondata == 'add') {
                    return $resultPageFactory->setPath('*/*/form');
                }
                if ($buttondata == 'close') {
                    return $resultPageFactory->setPath('*/*/index');
                }
                $id = $model->getId();
                return $resultPageFactory->setPath('*/*/form', ['id' => $id]);
            }
        } catch (\Exception $e) {
            $this->messageManager->addErrorMessage($e, __("We can't submit your request, Please try again."));
        }
        return $resultPageFactory->setPath('*/*/index');
    }
}
```

Step 15: Create the Form.php file in Controller folder:

A Form.php file in a custom controller could potentially be responsible for rendering a form for creating or updating and processing a form submission, saving data to the database, or performing some other action related to form management.

app/code/Emb/Module/Controller/Adminhtml/Sample/Form.php



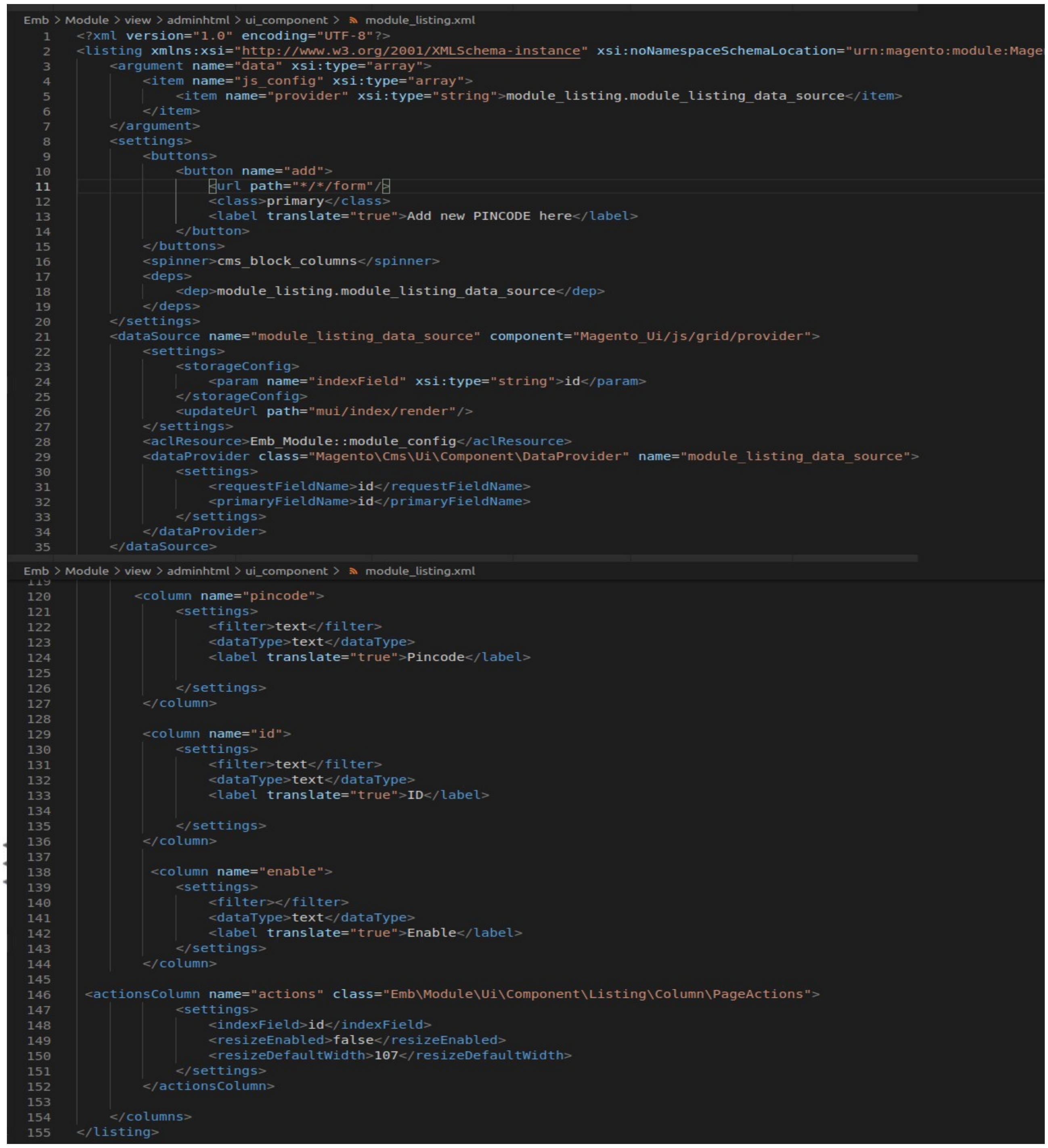
Emb > Module > Controller > Adminhtml > Sample > Form.php

```php
<?php

namespace Emb\Module\Controller\Adminhtml\Sample;

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

class Form extends Action
{
    /**
     * Authorization level of a basic admin session
     *
     * @see _isAllowed()
     */
    const ACTION_RESOURCE='Emb_Module::module';
    /**
     * Core registry
     *
     * @var Registry
     */
    protected $coreRegistry;
    /**
     * Result PageFactory
     *
     * @var PageFactory
     *
     */
    protected $resultPageFactory;
    /**
     * @var ForwardFactory
     */
    protected $resultForwardFactoty;
    /**
     * @param Registry $registry
     * @param PageFactory $resultPageFactory
     * @param ForwardFactory $resultForwardFactoty
     * @param Context $context
     */
    public function __construct(
        Registry $registry,
        PageFactory $resultPageFactory,
        ForwardFactory $resultForwardFactoty,
        Context $context
    ) {
            $this->coreRegistry =$registry;
            $this->resultPageFactory =$resultPageFactory;

            $this->resultForwardFactory =$resultForwardFactoty;
            parent::__construct($context);
    }
          /**
           * @return \Magento\Framework\View\Result\Page
           */
    public function execute()
    {
        $resultPage=$this->resultPageFactory->create();
        $resultPage->addBreadcrumb(__('Pincode'), __('Pincode'));
        $resultPage->getConfig()->getTitle()->prepend(__('Pincode'));
        return $resultPage;
    }
}
```

Step 16: Create the module_listing.xml:

module_isting.xml file is a UI Component file that is used to define the structure and behavior of a listing page for a specific entity type.This file is typically located n the view/adminhtml/ui component directory of a custom module or extension and is used to configure the appearance and functionality of the listing page in the Magento Admin Panel.

app/code/Emb/Module/view/adminhtml/ui_component/module_listing.xml



Emb > Module > view > adminhtml > ui_component > module_listing.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<listing xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:module:Mage
    <argument name="data" xsi:type="array">
        <item name="js_config" xsi:type="array">
            <item name="provider" xsi:type="string">module_listing.module_listing_data_source</item>
        </item>
    </argument>
    <settings>
        <buttons>
            <button name="add">
                <url path="*/*/form"/>
                <class>primary</class>
                <label translate="true">Add new PINCODE here</label>
            </button>
        </buttons>
        <spinner>cms_block_columns</spinner>
        <deps>
            <dep>module_listing.module_listing_data_source</dep>
        </deps>
    </settings>
    <dataSource name="module_listing_data_source" component="Magento_Ui/js/grid/provider">
        <settings>
            <storageConfig>
                <param name="indexField" xsi:type="string">id</param>
            </storageConfig>
            <updateUrl path="mui/index/render"/>
        </settings>
        <aclResource>Emb_Module::module_config</aclResource>
        <dataProvider class="Magento\Cms\Ui\Component\DataProvider" name="module_listing_data_source">
            <settings>
                <requestFieldName>id</requestFieldName>
                <primaryFieldName>id</primaryFieldName>
            </settings>
        </dataProvider>
    </dataSource>
```

Emb > Module > view > adminhtml > ui_component > module_listing.xml

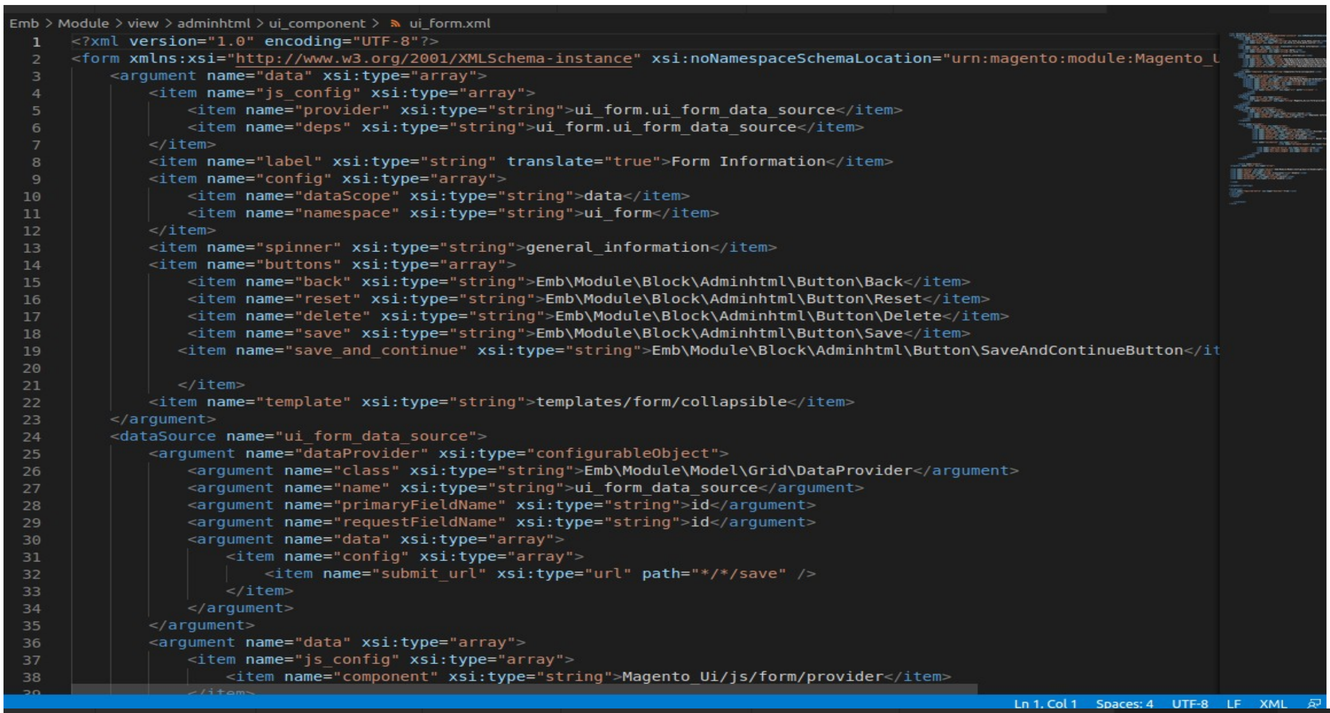
```xml
           <column name="pincode">
                <settings>
                    <filter>text</filter>
                    <dataType>text</dataType>
                    <label translate="true">Pincode</label>

                </settings>
            </column>

            <column name="id">
                <settings>
                    <filter>text</filter>
                    <dataType>text</dataType>
                    <label translate="true">ID</label>

                </settings>
            </column>

             <column name="enable">
                <settings>
                    <filter></filter>
                    <dataType>text</dataType>
                    <label translate="true">Enable</label>
                </settings>
            </column>

     <actionsColumn name="actions" class="Emb\Module\Ui\Component\Listing\Column\PageActions">
                <settings>
                    <indexField>id</indexField>
                    <resizeEnabled>false</resizeEnabled>
                    <resizeDefaultWidth>107</resizeDefaultWidth>
                </settings>
            </actionsColumn>

        </columns>
</listing>
```

i

Step 17: Create the ui_form.xml:

The form.xml file defines the structure of the form page by specifying the fields and input elements that should be displayed to the user. It also defines the data sources that should be used to populate the form fields, such as database tables or API endpoints.

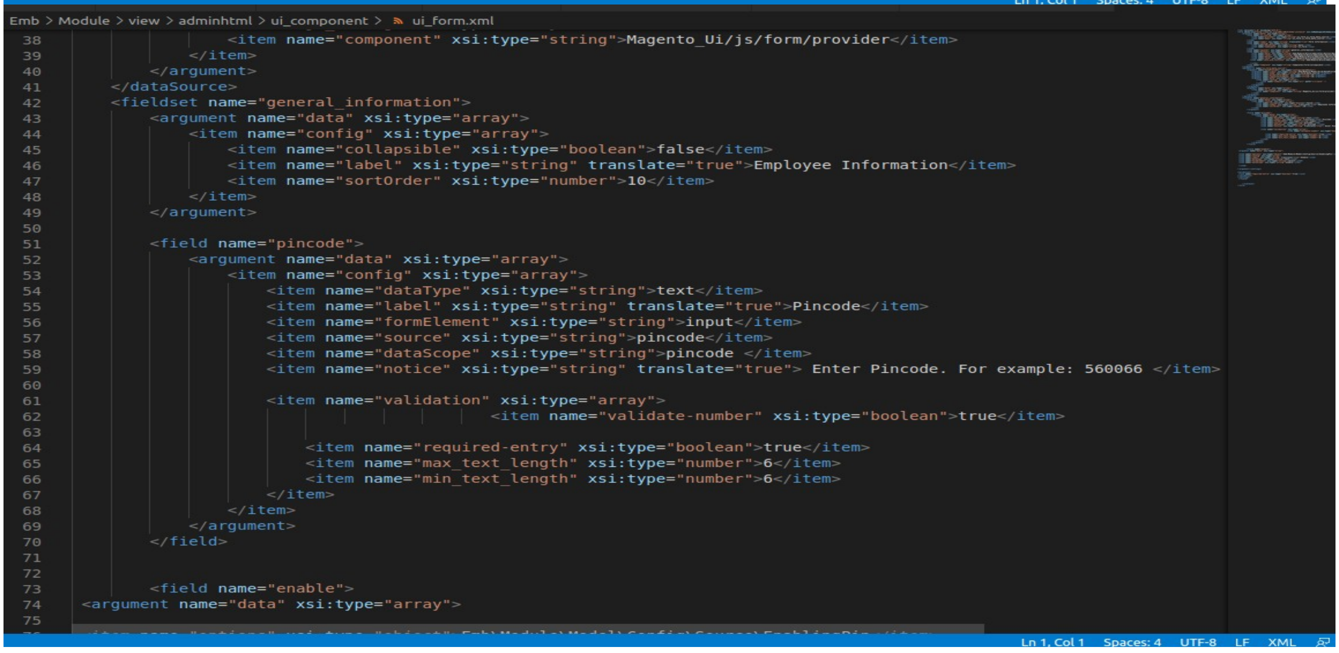
app/code/Emb/Module/view/adminhtml/ui_component/ui_form.xml



Emb > Module > view > adminhtml > ui_component > ui_form.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<form xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_U
    <argument name="data" xsi:type="array">
        <item name="js_config" xsi:type="array">
            <item name="provider" xsi:type="string">ui_form.ui_form_data_source</item>
            <item name="deps" xsi:type="string">ui_form.ui_form_data_source</item>
        </item>
        <item name="label" xsi:type="string" translate="true">Form Information</item>
        <item name="config" xsi:type="array">
            <item name="dataScope" xsi:type="string">data</item>
            <item name="namespace" xsi:type="string">ui_form</item>
        </item>
        <item name="spinner" xsi:type="string">general_information</item>
        <item name="buttons" xsi:type="array">
            <item name="back" xsi:type="string">Emb\Module\Block\Adminhtml\Button\Back</item>
            <item name="reset" xsi:type="string">Emb\Module\Block\Adminhtml\Button\Reset</item>
            <item name="delete" xsi:type="string">Emb\Module\Block\Adminhtml\Button\Delete</item>
            <item name="save" xsi:type="string">Emb\Module\Block\Adminhtml\Button\Save</item>
           <item name="save_and_continue" xsi:type="string">Emb\Module\Block\Adminhtml\Button\SaveAndContinueButton</it

           </item>
        <item name="template" xsi:type="string">templates/form/collapsible</item>
    </argument>
    <dataSource name="ui_form_data_source">
        <argument name="dataProvider" xsi:type="configurableObject">
            <argument name="class" xsi:type="string">Emb\Module\Model\Grid\DataProvider</argument>
            <argument name="name" xsi:type="string">ui_form_data_source</argument>
            <argument name="primaryFieldName" xsi:type="string">id</argument>
            <argument name="requestFieldName" xsi:type="string">id</argument>
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="submit_url" xsi:type="url" path="*/*/save" />
                </item>
            </argument>
        </argument>
        <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>
```

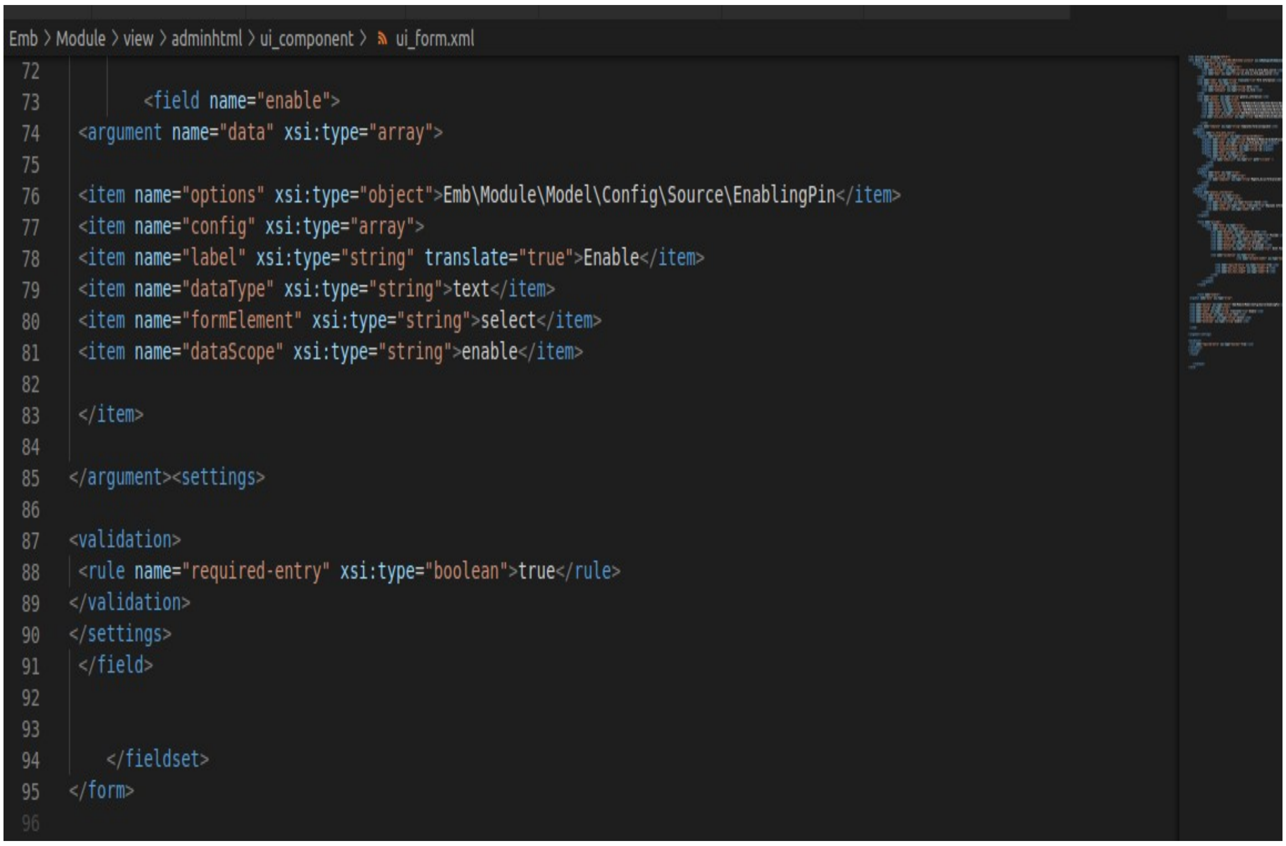
Ln 1, Col 1
Spaces: 4
UTF-8
LF
XML



Emb > Module > view > adminhtml > ui_component > ui_form.xml

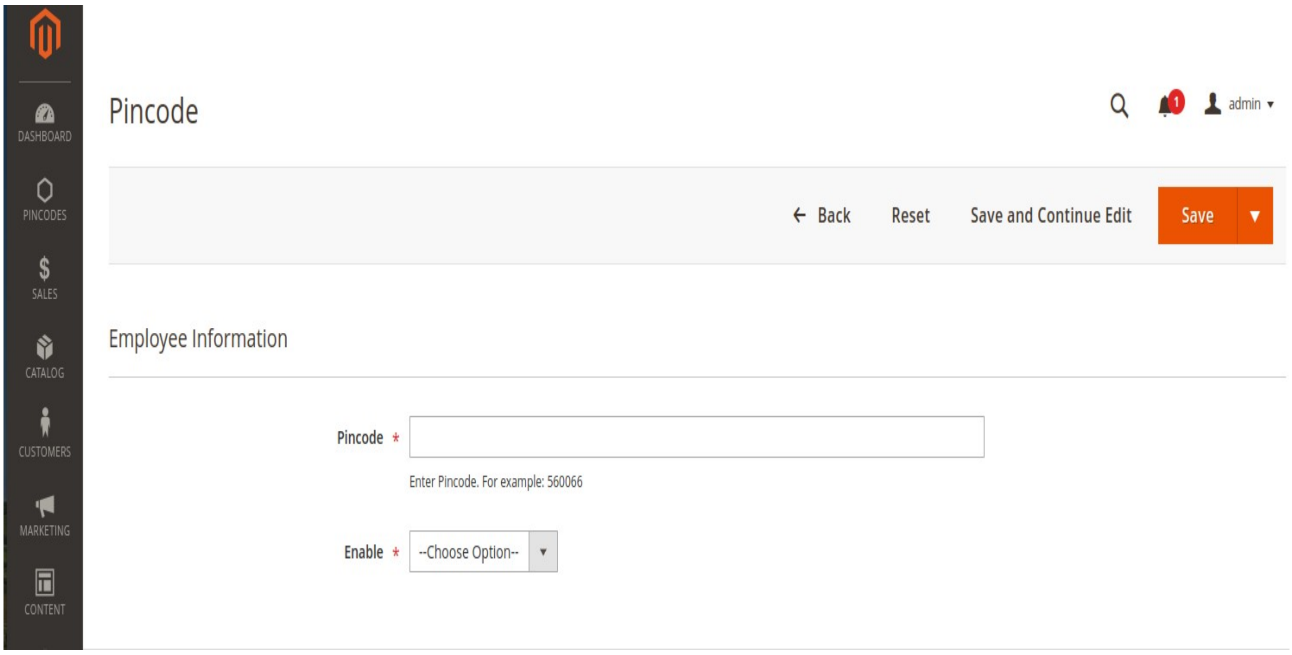
```xml
                <item name="component" xsi:type="string">Magento_Ui/js/form/provider</item>
            </item>
        </argument>
    </dataSource>
    <fieldset name="general_information">
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="collapsible" xsi:type="boolean">false</item>
                <item name="label" xsi:type="string" translate="true">Employee Information</item>
                <item name="sortOrder" xsi:type="number">10</item>
            </item>
        </argument>

        <field name="pincode">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="dataType" xsi:type="string">text</item>
                    <item name="label" xsi:type="string" translate="true">Pincode</item>
                    <item name="formElement" xsi:type="string">input</item>
                    <item name="source" xsi:type="string">pincode</item>
                    <item name="dataScope" xsi:type="string">pincode </item>
                    <item name="notice" xsi:type="string" translate="true"> Enter Pincode. For example: 560066 </item>

                    <item name="validation" xsi:type="array">
                                                <item name="validate-number" xsi:type="boolean">true</item>

                        <item name="required-entry" xsi:type="boolean">true</item>
                        <item name="max_text_length" xsi:type="number">6</item>
                        <item name="min_text_length" xsi:type="number">6</item>
                    </item>
                </item>
            </argument>
        </field>


        <field name="enable">
 <argument name="data" xsi:type="array">

```

Ln 1, Col 1
Spaces: 4
UTF-8
LF
XML



Emb › Module › view › adminhtml › ui_component › ui_form.xml

```
            <field name="enable">
    <argument name="data" xsi:type="array">

    <item name="options" xsi:type="object">Emb\Module\Model\Config\Source\EnablingPin</item>
    <item name="config" xsi:type="array">
    <item name="label" xsi:type="string" translate="true">Enable</item>
    <item name="dataType" xsi:type="string">text</item>
    <item name="formElement" xsi:type="string">select</item>
    <item name="dataScope" xsi:type="string">enable</item>

    </item>

   </argument><settings>

   <validation>
    <rule name="required-entry" xsi:type="boolean">true</rule>
   </validation>
   </settings>
    </field>


       </fieldset>
   </form>
```



Pincode

Employee Information

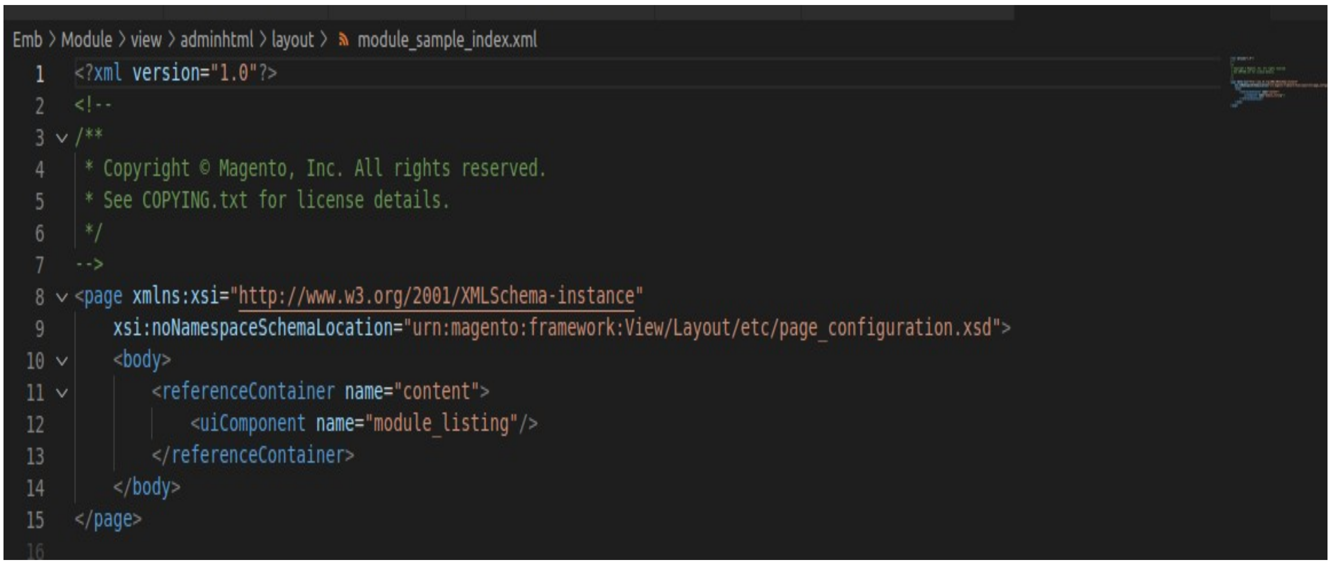
Pincode *

Enter Pincode. For example: 560066

Enable * --Choose Option--

Step 19: Create the module_sample_index.xml in view-adminhtml-layout folder: The index.xml file can define the layout of the blocks on the dashboard page, such as the position, size, and alignment of each block. It can also specify the content that should be displayed in each block.

app/code/Emb/Module/view/adminhtml/layout/module_sample_index.xml



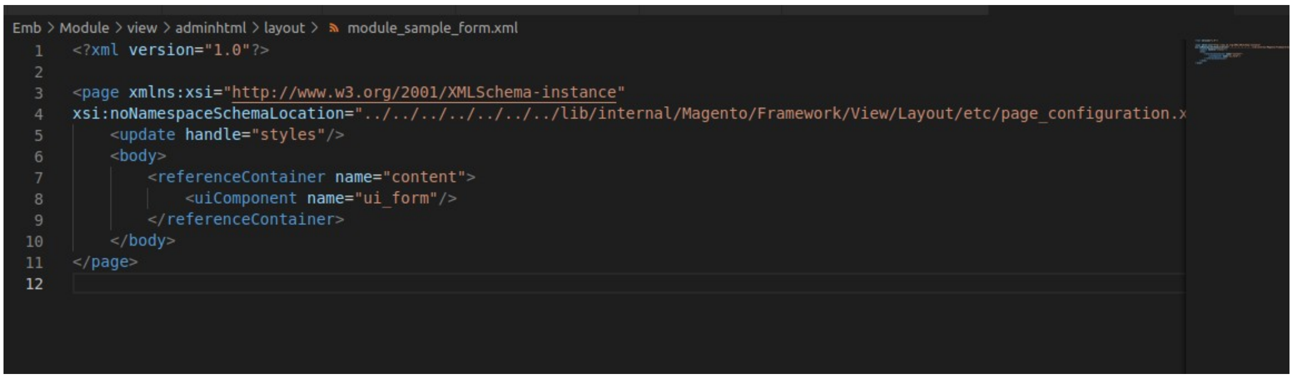
Emb > Module > view > adminhtml > layout > module_sample_index.xml

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

Step 20: Create the module_sample_form.xml in view-adminhtml-layout folder:

The form.xml file can define the layout of the blocks on the dashboard page, such as the position, size, and alignment of each block. It can also specify the content that should be displayed in each block, such as Pincode number, Enabiling options.The form.xml file include instructions for adding custom buttons or actions to the form page, or for modifying existing buttons or actions to add additional functionality or styling.

app/code/Emb/Module/view/adminhtml/layout/module_sample_form.xml



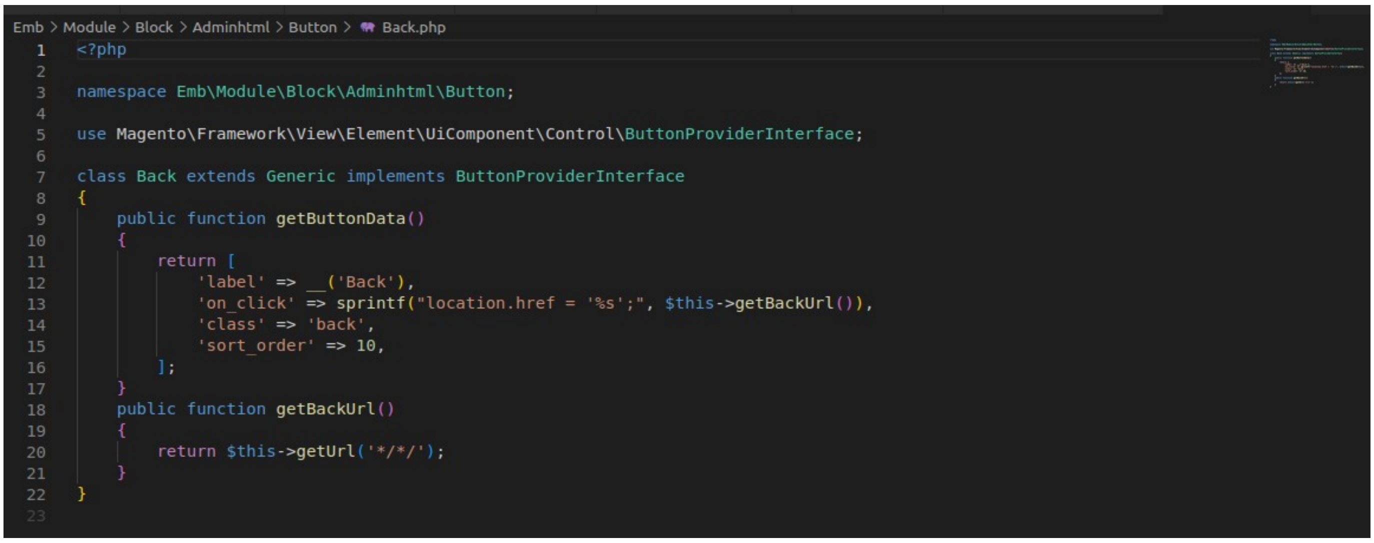
Emb > Module > view > adminhtml > layout > module_sample_form.xml

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

<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../../../../lib/internal/Magento/Framework/View/Layout/etc/page_configuration.x
    <update handle="styles"/>
    <body>
        <referenceContainer name="content">
            <uiComponent name="ui_form"/>
        </referenceContainer>
    </body>
</page>
```

2.2 ADDING THE BUTTONS:

1. Back.php: In Magento the back button is used to navigate to the previous page in the users browsing history.

app/code/Emb/Module/Block/Adminhtml/Button/Back.php

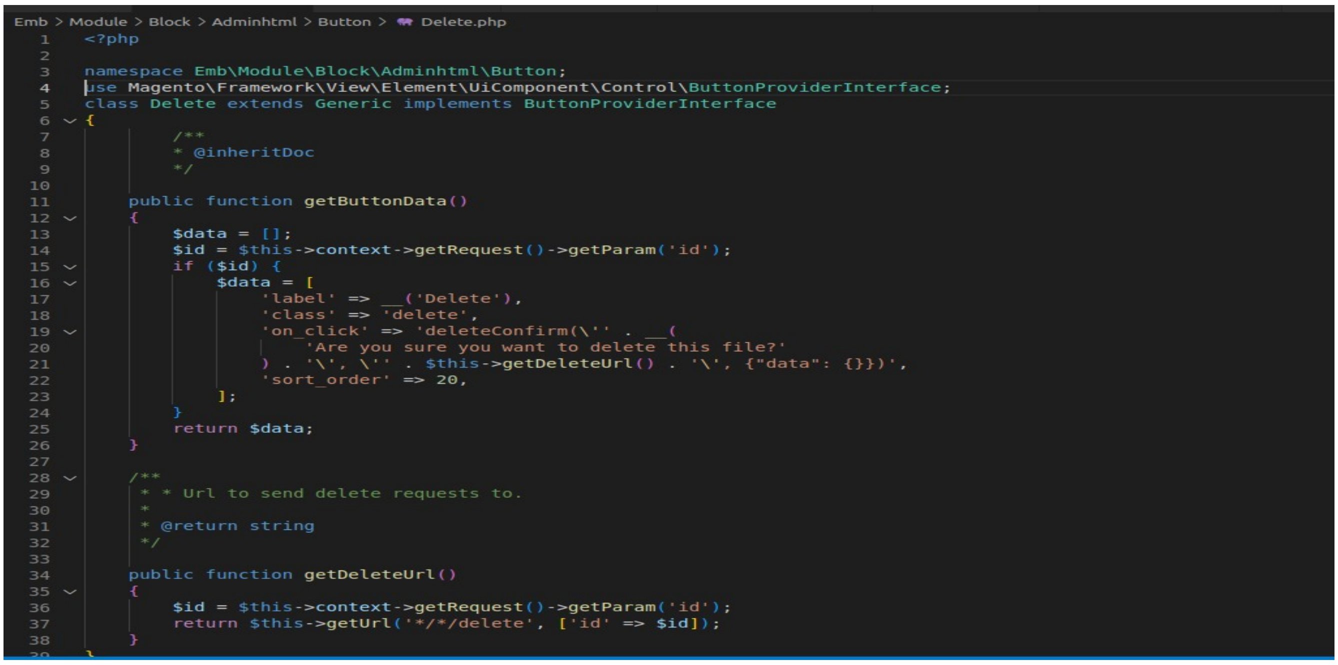


Emb > Module > Block > Adminhtml > Button > Back.php

```php
<?php

namespace Emb\Module\Block\Adminhtml\Button;

use Magento\Framework\View\Element\UiComponent\Control\ButtonProviderInterface;

class Back extends Generic implements ButtonProviderInterface
{
    public function getButtonData()
    {
        return [
            'label' => __('Back'),
            'on_click' => sprintf("location.href = '%s';", $this->getBackUrl()),
            'class' => 'back',
            'sort_order' => 10,
        ];
    }
    public function getBackUrl()
    {
        return $this->getUrl('*/*/');
    }
}
```

2. Delete.php: The delete button is used to delete the details of the storelocator in which the user can add the other details in the store form.

app/code/Emb/Module/Block/Adminhtml/Button/Delete.php

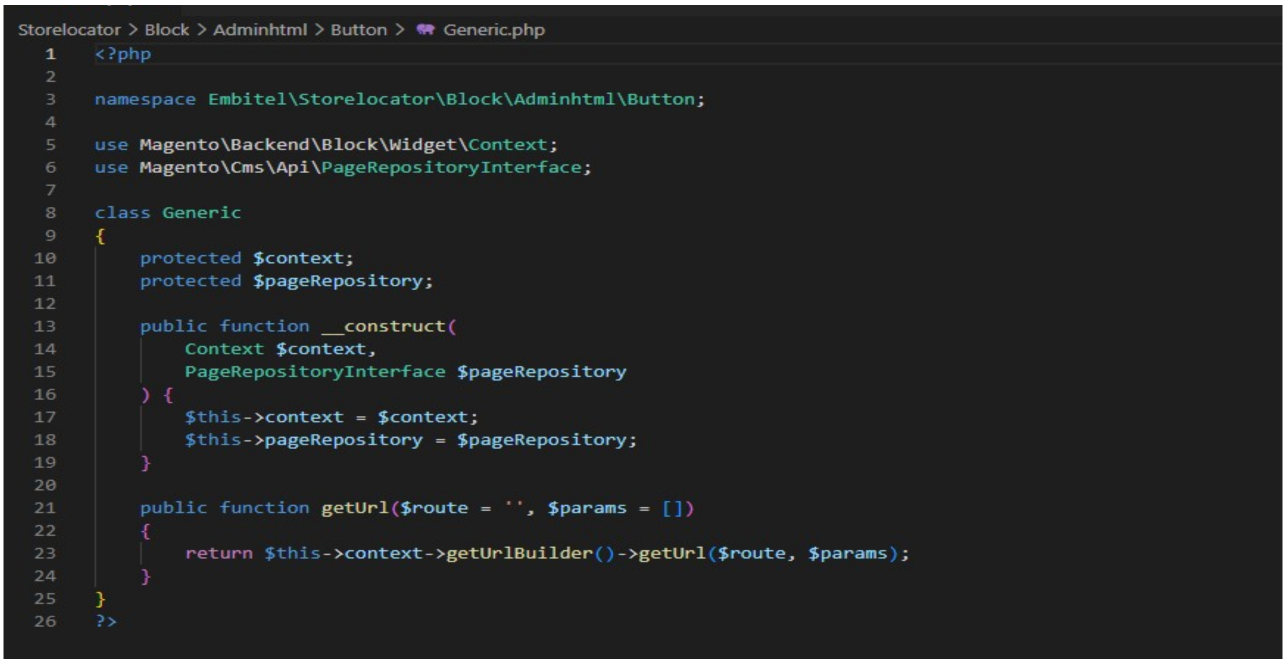


Emb > Module > Block > Adminhtml > Button > Delete.php

```php
<?php

namespace Emb\Module\Block\Adminhtml\Button;
use Magento\Framework\View\Element\UiComponent\Control\ButtonProviderInterface;
class Delete extends Generic implements ButtonProviderInterface
{
        /**
        * @inheritDoc
        */

    public function getButtonData()
    {
        $data = [];
        $id = $this->context->getRequest()->getParam('id');
        if ($id) {
            $data = [
                'label' => __('Delete'),
                'class' => 'delete',
                'on_click' => 'deleteConfirm(\'' . __(
                    'Are you sure you want to delete this file?'
                ) . '\', \'' . $this->getDeleteUrl() . '\', {"data": {}})',
                'sort_order' => 20,
            ];
        }
        return $data;
    }

    /**
     * * Url to send delete requests to.
     *
     * @return string
     */

    public function getDeleteUrl()
    {
        $id = $this->context->getRequest()->getParam('id');
        return $this->getUrl('*/*/delete', ['id' => $id]);
    }
```

3.Generic.php:

app/code/Emb/Module/Block/Adminhtml/Button/Generic.php

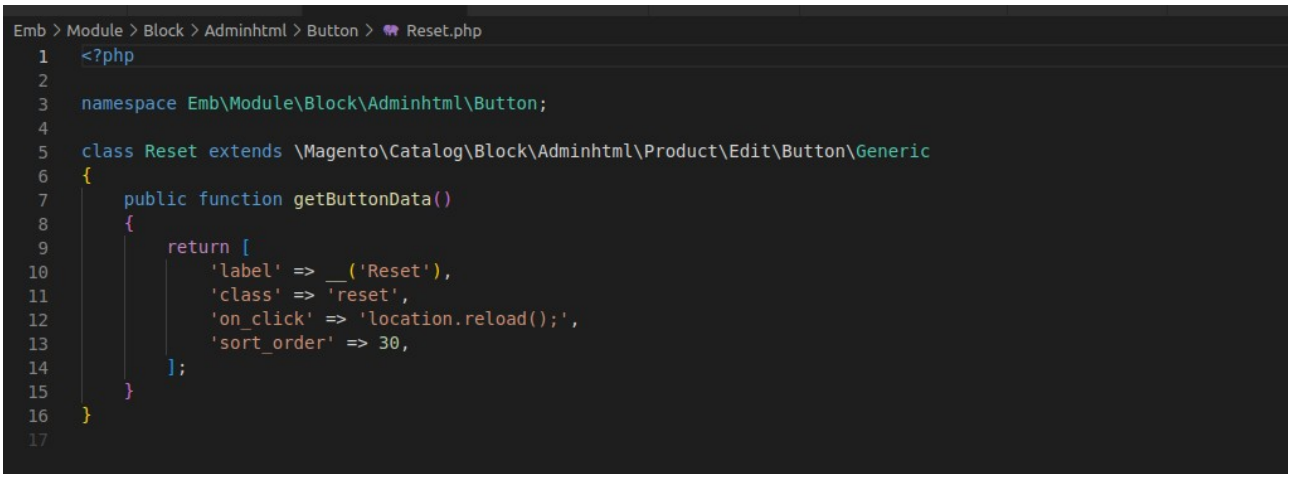


Storelocator > Block > Adminhtml > Button > Generic.php

```php
<?php

namespace Embitel\Storelocator\Block\Adminhtml\Button;

use Magento\Backend\Block\Widget\Context;
use Magento\Cms\Api\PageRepositoryInterface;

class Generic
{
    protected $context;
    protected $pageRepository;

    public function __construct(
        Context $context,
        PageRepositoryInterface $pageRepository
    ) {
        $this->context = $context;
        $this->pageRepository = $pageRepository;
    }

    public function getUrl($route = '', $params = [])
    {
        return $this->context->getUrlBuilder()->getUrl($route, $params);
    }
}
?>
```

4. Reset.php: It is typically used to undo any changes made to the form or configuration settings and restore the default values and found in the same page of save button in which the user can change various settings.

app/code/Emb/Module/Block/Adminhtml/Button/Reset.php

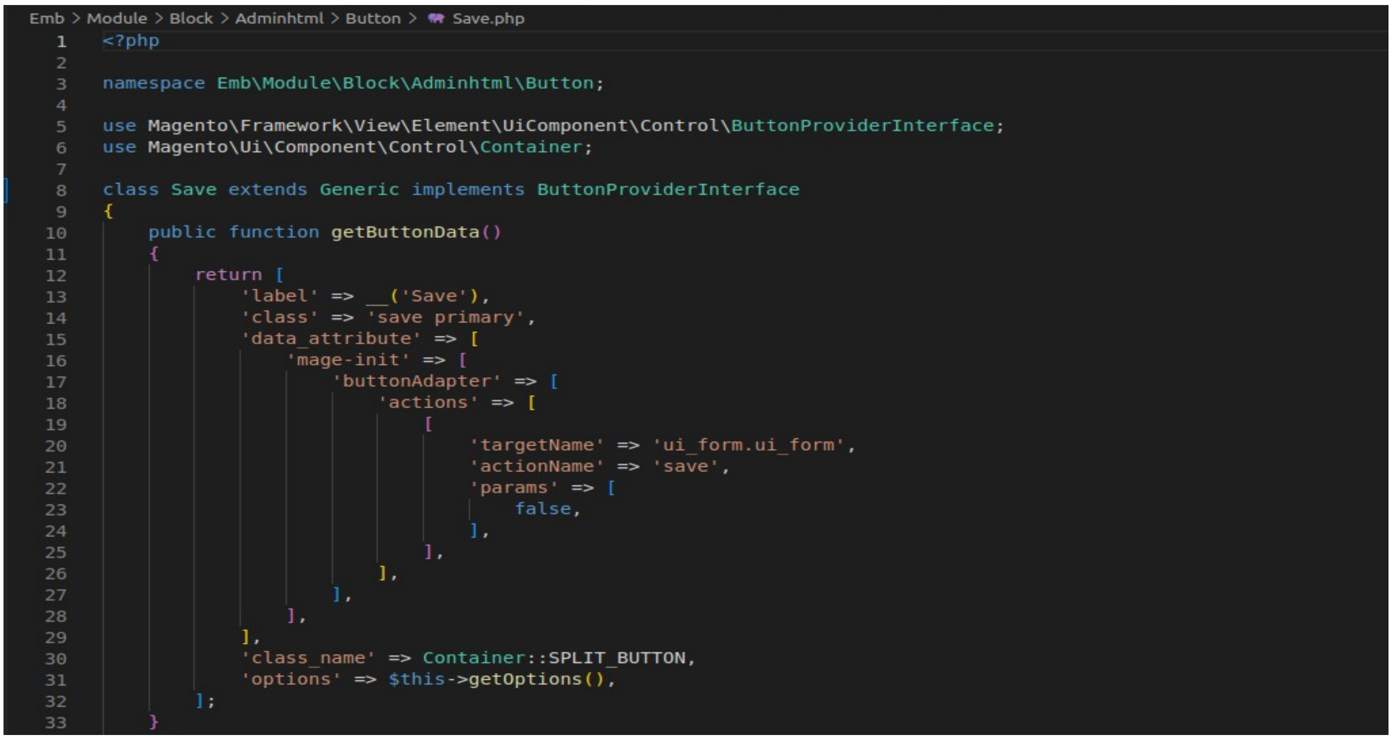


Emb > Module > Block > Adminhtml > Button > Reset.php

```php
<?php

namespace Emb\Module\Block\Adminhtml\Button;

class Reset extends \Magento\Catalog\Block\Adminhtml\Product\Edit\Button\Generic
{
    public function getButtonData()
    {
        return [
            'label' => __('Reset'),
            'class' => 'reset',
            'on_click' => 'location.reload();',
            'sort_order' => 30,
        ];
    }
}

```

5. Save.php: The save.php is used to apply the changes made to a form or configuration changes and update the corresponding entity in the system and the user can make changes.

app/code/Emb/Module/Block/Adminhtml/Button/Save.php



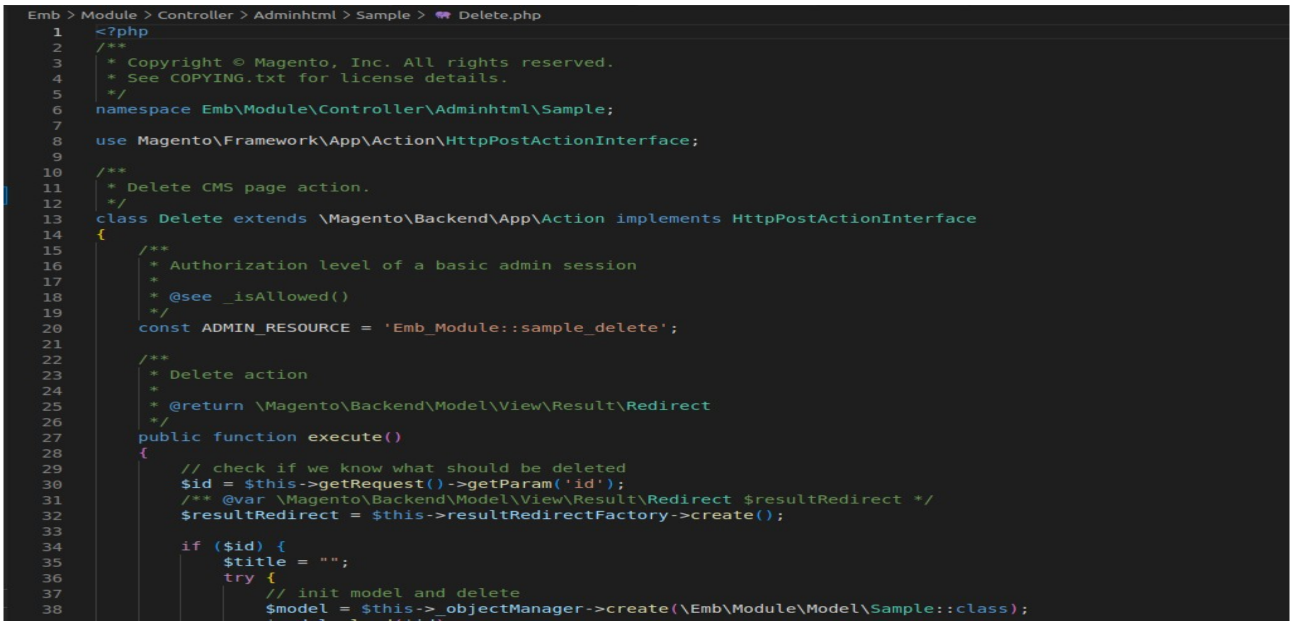
Emb > Module > Block > Adminhtml > Button > Save.php

```php
<?php

namespace Emb\Module\Block\Adminhtml\Button;

use Magento\Framework\View\Element\UiComponent\Control\ButtonProviderInterface;
use Magento\Ui\Component\Control\Container;

class Save extends Generic implements ButtonProviderInterface
{
    public function getButtonData()
    {
        return [
            'label' => __('Save'),
            'class' => 'save primary',
            'data_attribute' => [
                'mage-init' => [
                    'buttonAdapter' => [
                        'actions' => [
                            [
                                'targetName' => 'ui_form.ui_form',
                                'actionName' => 'save',
                                'params' => [
                                    false,
                                ],
                            ],
                        ],
                    ],
                ],
            ],
            'class_name' => Container::SPLIT_BUTTON,
            'options' => $this->getOptions(),
        ];
    }
```

2.3 ADDING UI COMPONENT:

1. Delete.php: It is the standard file name used in the adminhtml controller directory to handle the deletions of specific entity in which the customer can delete the details.

app/code/Emb/Module/Controller/Adminhtml/Sample/Delete.php

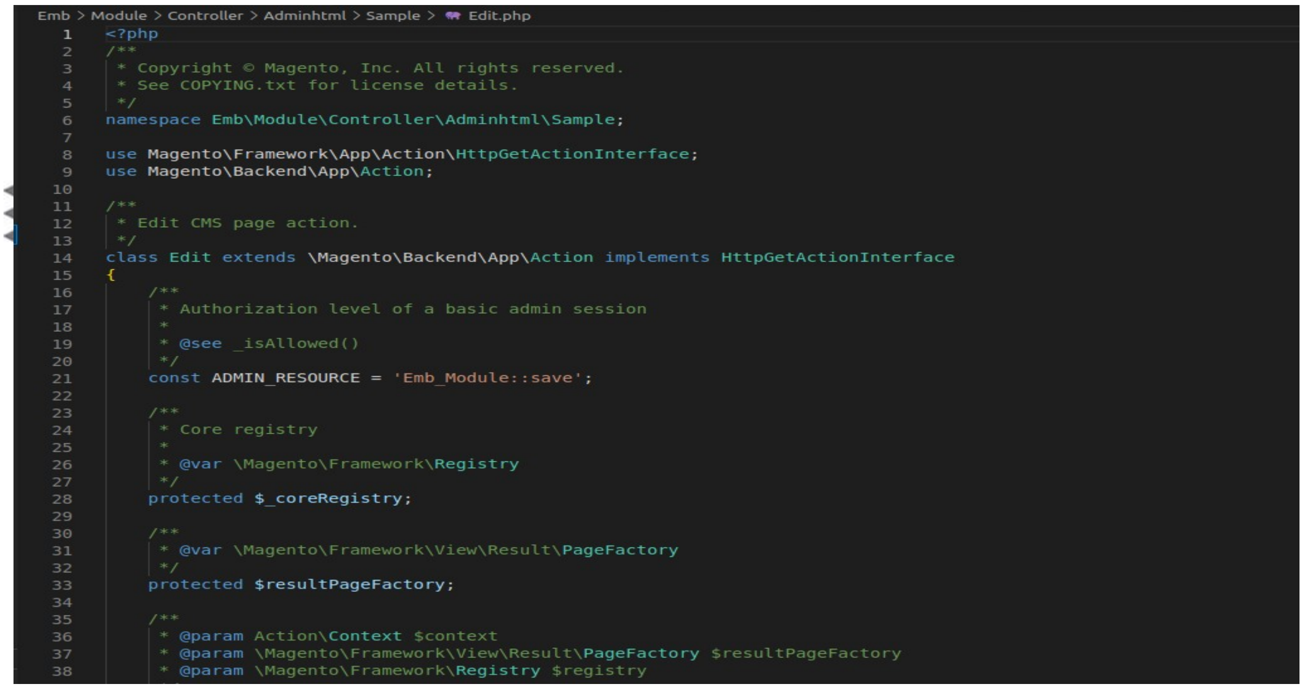


Emb > Module > Controller > Adminhtml > Sample > Delete.php

```php
<?php
/**
 * Copyright © Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Emb\Module\Controller\Adminhtml\Sample;

use Magento\Framework\App\Action\HttpPostActionInterface;

/**
 * Delete CMS page action.
 */
class Delete extends \Magento\Backend\App\Action implements HttpPostActionInterface
{
    /**
     * Authorization level of a basic admin session
     *
     * @see _isAllowed()
     */
    const ADMIN_RESOURCE = 'Emb_Module::sample_delete';

    /**
     * Delete action
     *
     * @return \Magento\Backend\Model\View\Result\Redirect
     */
    public function execute()
    {
        // check if we know what should be deleted
        $id = $this->getRequest()->getParam('id');
        /** @var \Magento\Backend\Model\View\Result\Redirect $resultRedirect */
        $resultRedirect = $this->resultRedirectFactory->create();

        if ($id) {
            $title = "";
            try {
                // init model and delete
                $model = $this->_objectManager->create(\Emb\Module\Model\Sample::class);
```

2. Edit.php: It is the standard file name used in the adminhtml controller directory to handle the deletions of specific entity in which the customer can edit the details.

app/code/Emb/Module/Controller/Adminhtml/Sample/Edit.php

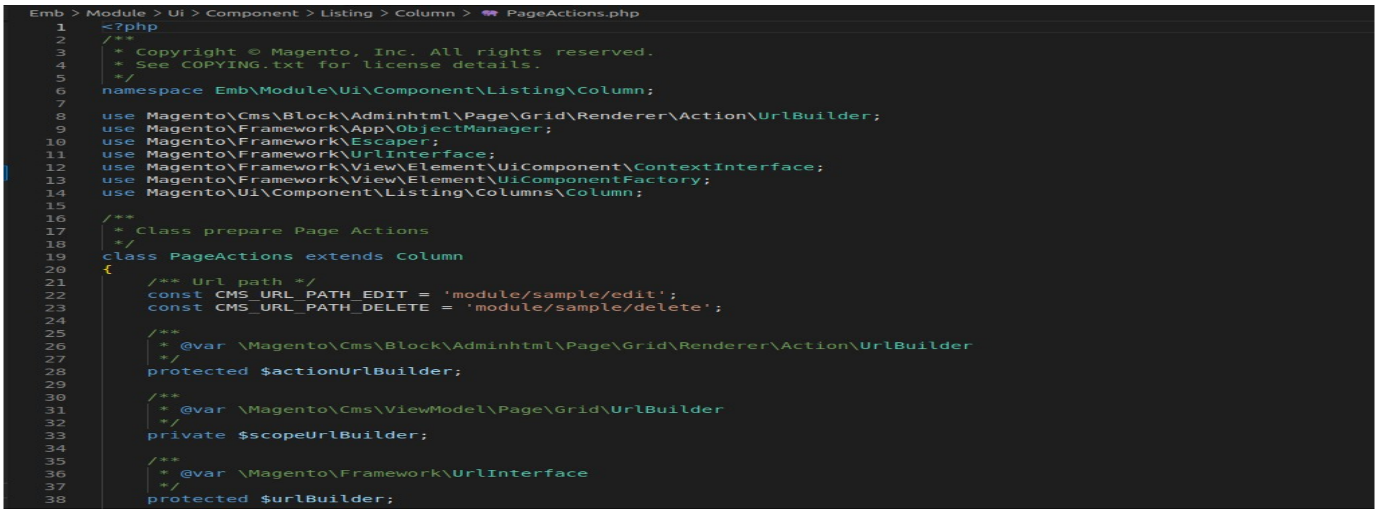


```
Emb > Module > Controller > Adminhtml > Sample > Edit.php
<?php
/**
 * Copyright © Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Emb\Module\Controller\Adminhtml\Sample;

use Magento\Framework\App\Action\HttpGetActionInterface;
use Magento\Backend\App\Action;

/**
 * Edit CMS page action.
 */
class Edit extends \Magento\Backend\App\Action implements HttpGetActionInterface
{
    /**
     * Authorization level of a basic admin session
     *
     * @see _isAllowed()
     */
    const ADMIN_RESOURCE = 'Emb_Module::save';

    /**
     * Core registry
     *
     * @var \Magento\Framework\Registry
     */
    protected $_coreRegistry;

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

    /**
     * @param Action\Context $context
     * @param \Magento\Framework\View\Result\PageFactory $resultPageFactory
     * @param \Magento\Framework\Registry $registry
```

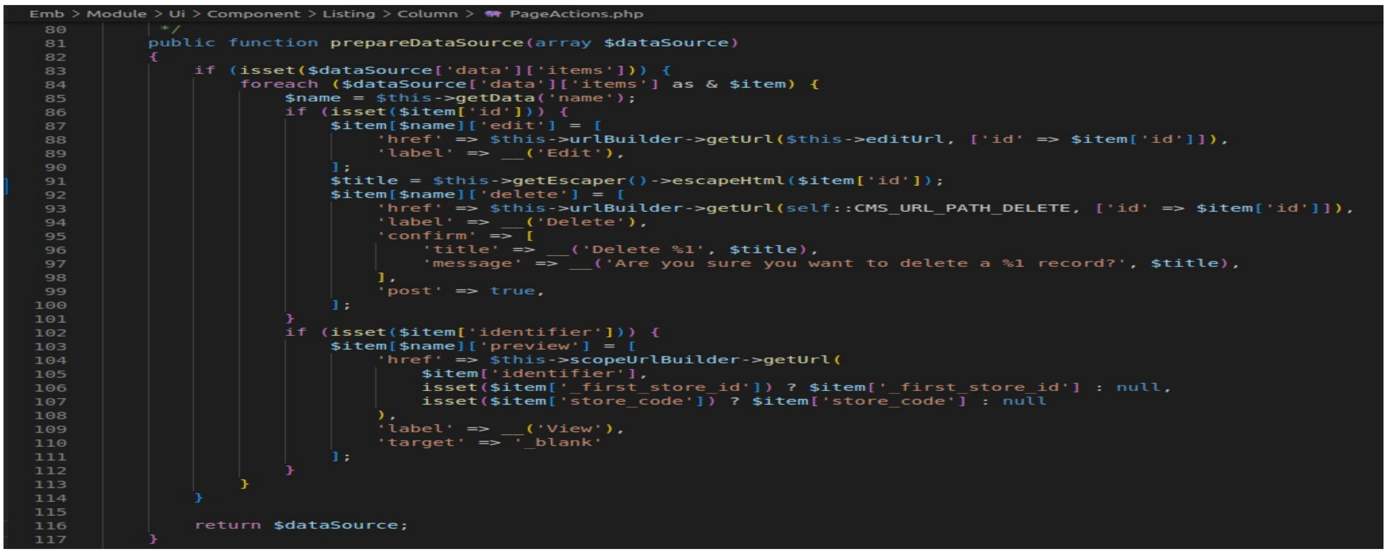
2.4 UI COMPONENT:

1. PageActions.php: It is a file in the UI component directory that is responsible for handling various actions triggered by the user interface. This file receives the request from UI components and execute the corresponding action based on the request parameter.

app/code/Emb/Module/Ui/Component/Listing/Column/PageActions.php



```
Emb > Module > Ui > Component > Listing > Column > PageActions.php
<?php
/**
 * Copyright © Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace Emb\Module\Ui\Component\Listing\Column;

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

/**
 * Class prepare Page Actions
 */
class PageActions extends Column
{
    /** Url path */
    const CMS_URL_PATH_EDIT = 'module/sample/edit';
    const CMS_URL_PATH_DELETE = 'module/sample/delete';

    /**
     * @var \Magento\Cms\Block\Adminhtml\Page\Grid\Renderer\Action\UrlBuilder
     */
    protected $actionUrlBuilder;

    /**
     * @var \Magento\Cms\ViewModel\Page\Grid\UrlBuilder
     */
    private $scopeUrlBuilder;

    /**
     * @var \Magento\Framework\UrlInterface
     */
    protected $urlBuilder;
```

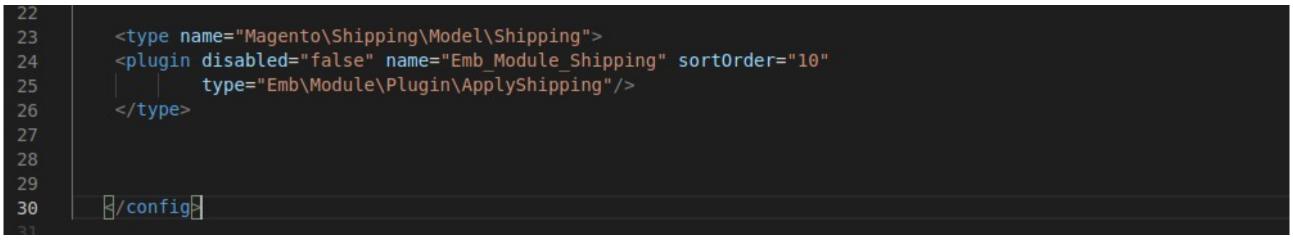

Emb > Module > Ui > Component > Listing > Column > PageActions.php

```php
     */
    public function prepareDataSource(array $dataSource)
    {
        if (isset($dataSource['data']['items'])) {
            foreach ($dataSource['data']['items'] as & $item) {
                $name = $this->getData('name');
                if (isset($item['id'])) {
                    $item[$name]['edit'] = [
                        'href' => $this->urlBuilder->getUrl($this->editUrl, ['id' => $item['id']]),
                        'label' => __('Edit'),
                    ];
                    $title = $this->getEscaper()->escapeHtml($item['id']);
                    $item[$name]['delete'] = [
                        'href' => $this->urlBuilder->getUrl(self::CMS_URL_PATH_DELETE, ['id' => $item['id']]),
                        'label' => __('Delete'),
                        'confirm' => [
                            'title' => __('Delete %1', $title),
                            'message' => __('Are you sure you want to delete a %1 record?', $title),
                        ],
                        'post' => true,
                    ];
                }
                if (isset($item['identifier'])) {
                    $item[$name]['preview'] = [
                        'href' => $this->scopeUrlBuilder->getUrl(
                            $item['identifier'],
                            isset($item['_first_store_id']) ? $item['_first_store_id'] : null,
                            isset($item['store_code']) ? $item['store_code'] : null
                        ),
                        'label' => __('View'),
                        'target' => '_blank'
                    ];
                }
            }
        }

        return $dataSource;
    }
```

2.5 Using PLUGIN validating:

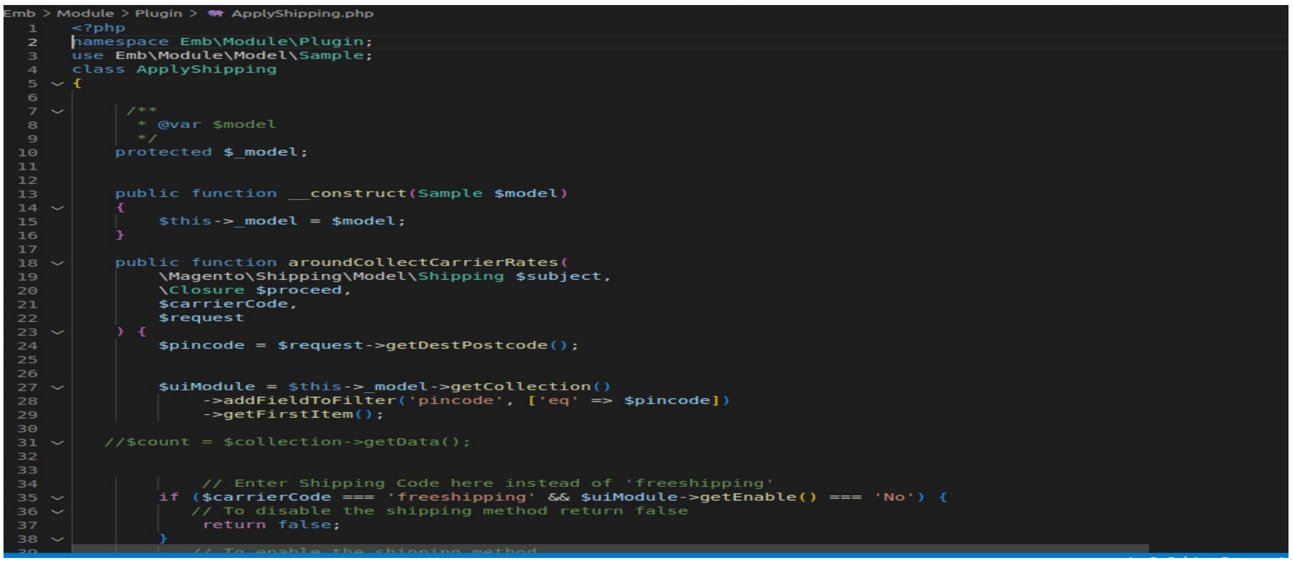
We are using plugin method for tha validation in di.xml file in the etc folder the type name "Magento\Shipping\Model\Shipping"

By using this we can enable free shipping for the postal codes which are storing in the database.



```xml

    <type name="Magento\Shipping\Model\Shipping">
    <plugin disabled="false" name="Emb_Module_Shipping" sortOrder="10"
            type="Emb\Module\Plugin\ApplyShipping"/>
    </type>



</config>
```

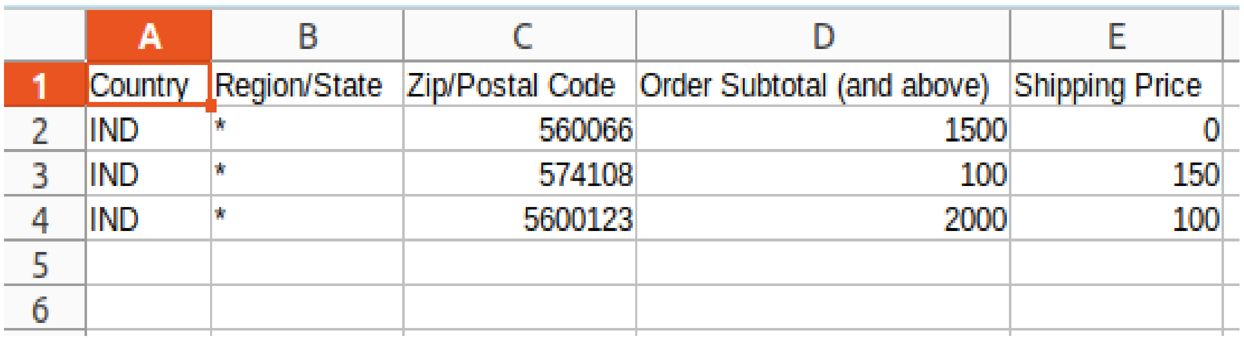
app/code/Emb/Module/Plugin/ApplyShipping.php


Emb > Module > Plugin > ApplyShipping.php

```php
<?php
namespace Emb\Module\Plugin;
use Emb\Module\Model\Sample;
class ApplyShipping
{

    /**
     * @var $model
     */
    protected $_model;


    public function __construct(Sample $model)
    {
        $this->_model = $model;
    }

    public function aroundCollectCarrierRates(
        \Magento\Shipping\Model\Shipping $subject,
        \Closure $proceed,
        $carrierCode,
        $request
    ) {
        $pincode = $request->getDestPostcode();


        $uiModule = $this->_model->getCollection()
            ->addFieldToFilter('pincode', ['eq' => $pincode])
            ->getFirstItem();

   //$count = $collection->getData();


            // Enter Shipping Code here instead of 'freeshipping'
        if ($carrierCode === 'freeshipping' && $uiModule->getEnable() === 'No') {
           // To disable the shipping method return false
            return false;
        }
           // To enable the shipping method
```

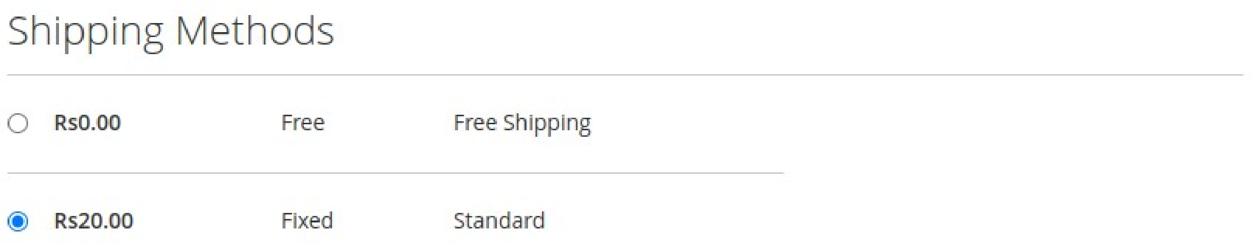
3.Enabiling Table rate Shipping:

In admin menu panel we first click on Stores->Configuration->Sales->Delivery methods->table rates after we have to add a csv.file in the table rate shipping.



	A	B	C	D	E
1	Country	Region/State	Zip/Postal Code	Order Subtotal (and above)	Shipping Price
2	IND	*	560066	1500	0
3	IND	*	574108	100	150
4	IND	*	5600123	2000	100
5					
6					

In the payment method based on pincodes different shipping methods will be generated

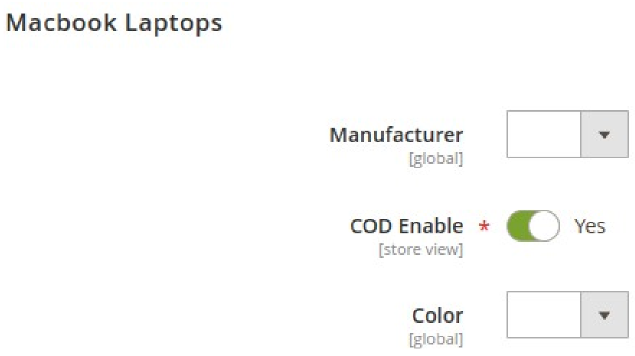


Shipping Methods

○	Rs0.00	Free	Free Shipping
●	Rs20.00	Fixed	Standard

4.COD Enable/Disable for customer & product:

For this COD Enabiling or Disabiling we have to create a Attribute called COD Enable and we select options as Yes/No. Based on this attribute we are providing the cod for the products and customers. For that enabiling or disabiling we used events and observer file.



Macbook Laptops

Manufacturer [global]

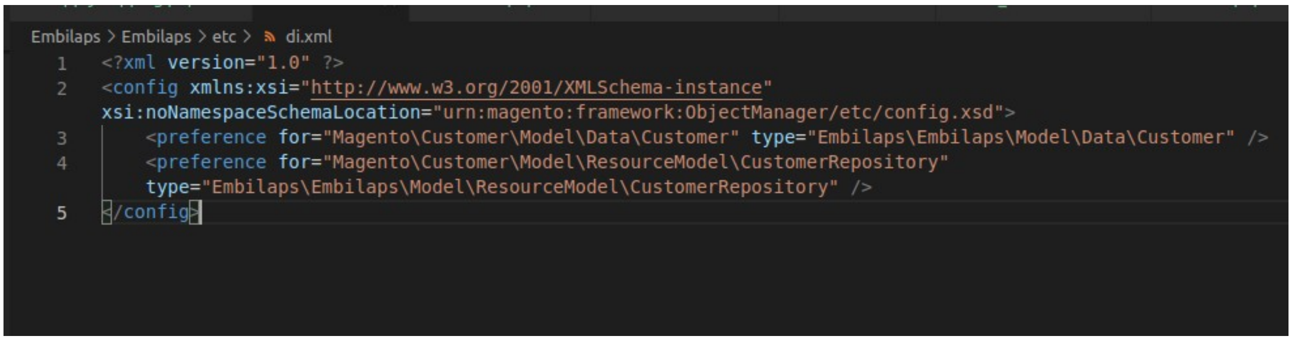
COD Enable * [store view] Yes

Color [global]

4.1. Creating the customer attribute:

We are using these following steps to

Embilaps/Embilaps/ets/di.xml

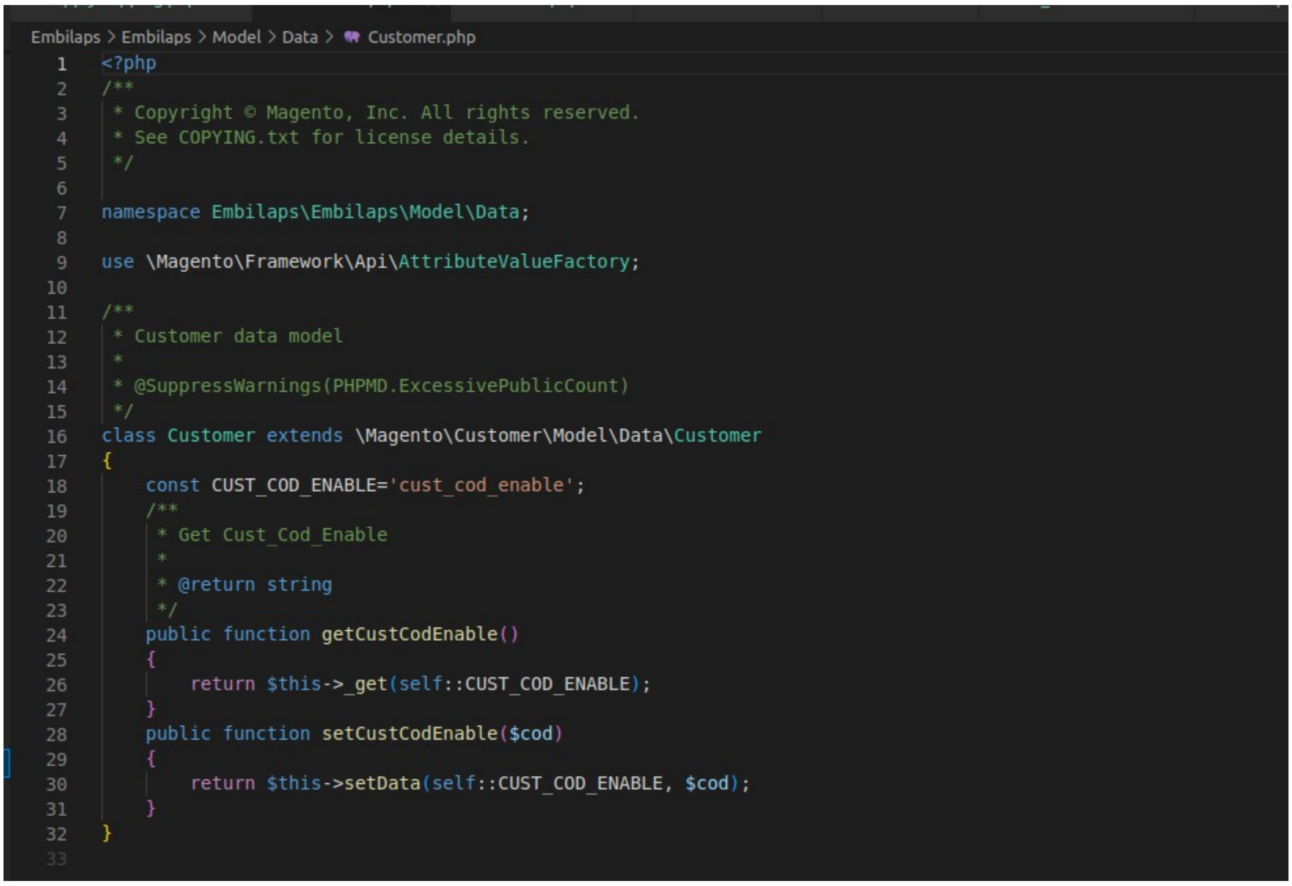


Embilaps > Embilaps > etc > di.xml

```xml
<?xml version="1.0" ?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <preference for="Magento\Customer\Model\Data\Customer" type="Embilaps\Embilaps\Model\Data\Customer" />
    <preference for="Magento\Customer\Model\ResourceModel\CustomerRepository"
    type="Embilaps\Embilaps\Model\ResourceModel\CustomerRepository" />
</config>
```

first we are overridding customer.php from Magento/Customer/Model/Data/Customer.php

Embilaps/Embilaps/Model/Data/Customer.php



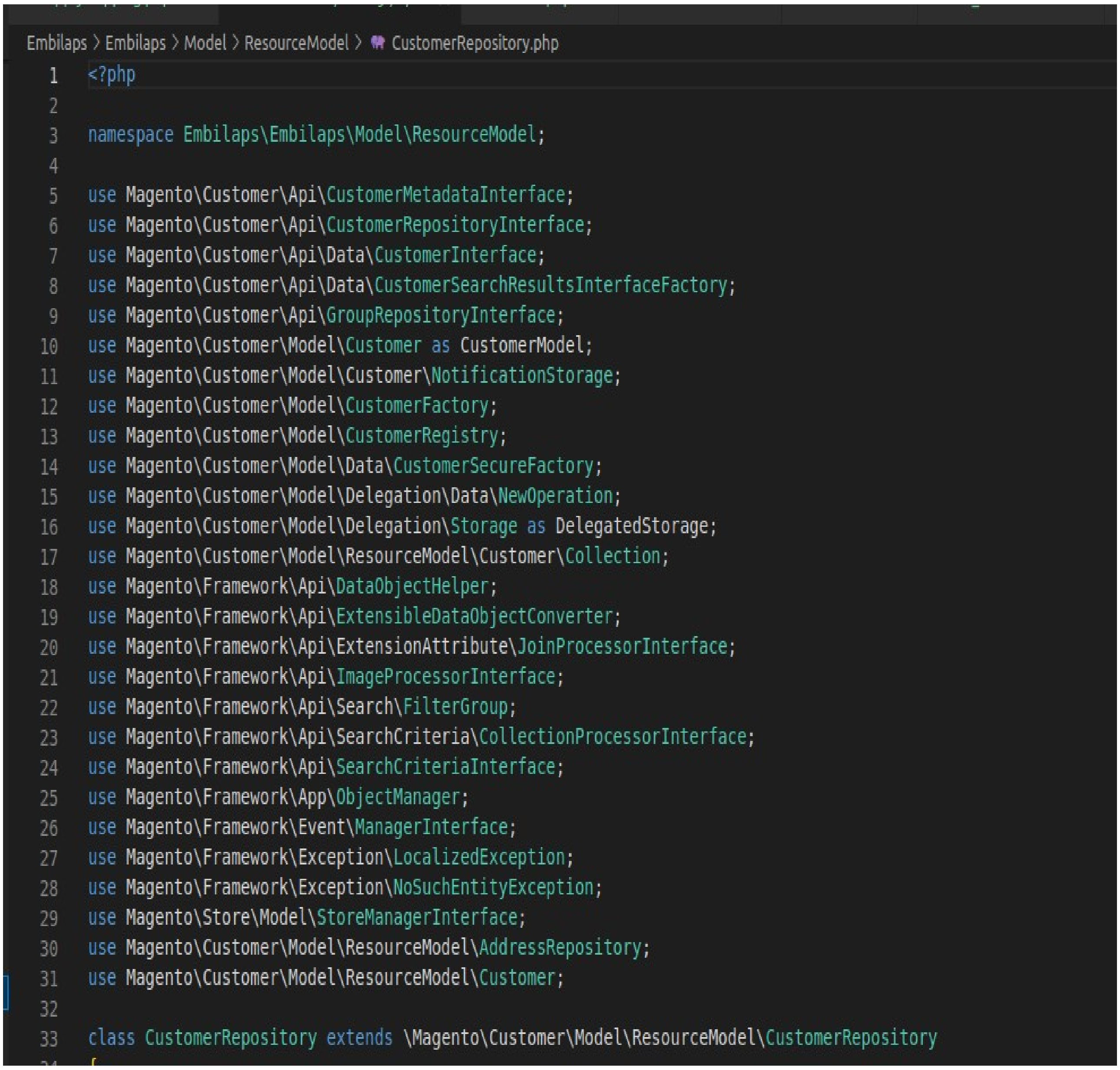
Embilaps > Embilaps > Model > Data > Customer.php

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

namespace Embilaps\Embilaps\Model\Data;

use \Magento\Framework\Api\AttributeValueFactory;

/**
 * Customer data model
 *
 * @SuppressWarnings(PHPMD.ExcessivePublicCount)
 */
class Customer extends \Magento\Customer\Model\Data\Customer
{
    const CUST_COD_ENABLE='cust_cod_enable';
    /**
     * Get Cust_Cod_Enable
     *
     * @return string
     */
    public function getCustCodEnable()
    {
        return $this->_get(self::CUST_COD_ENABLE);
    }
    public function setCustCodEnable($cod)
    {
        return $this->setData(self::CUST_COD_ENABLE, $cod);
    }
}
```

first we are overridding customer.php from

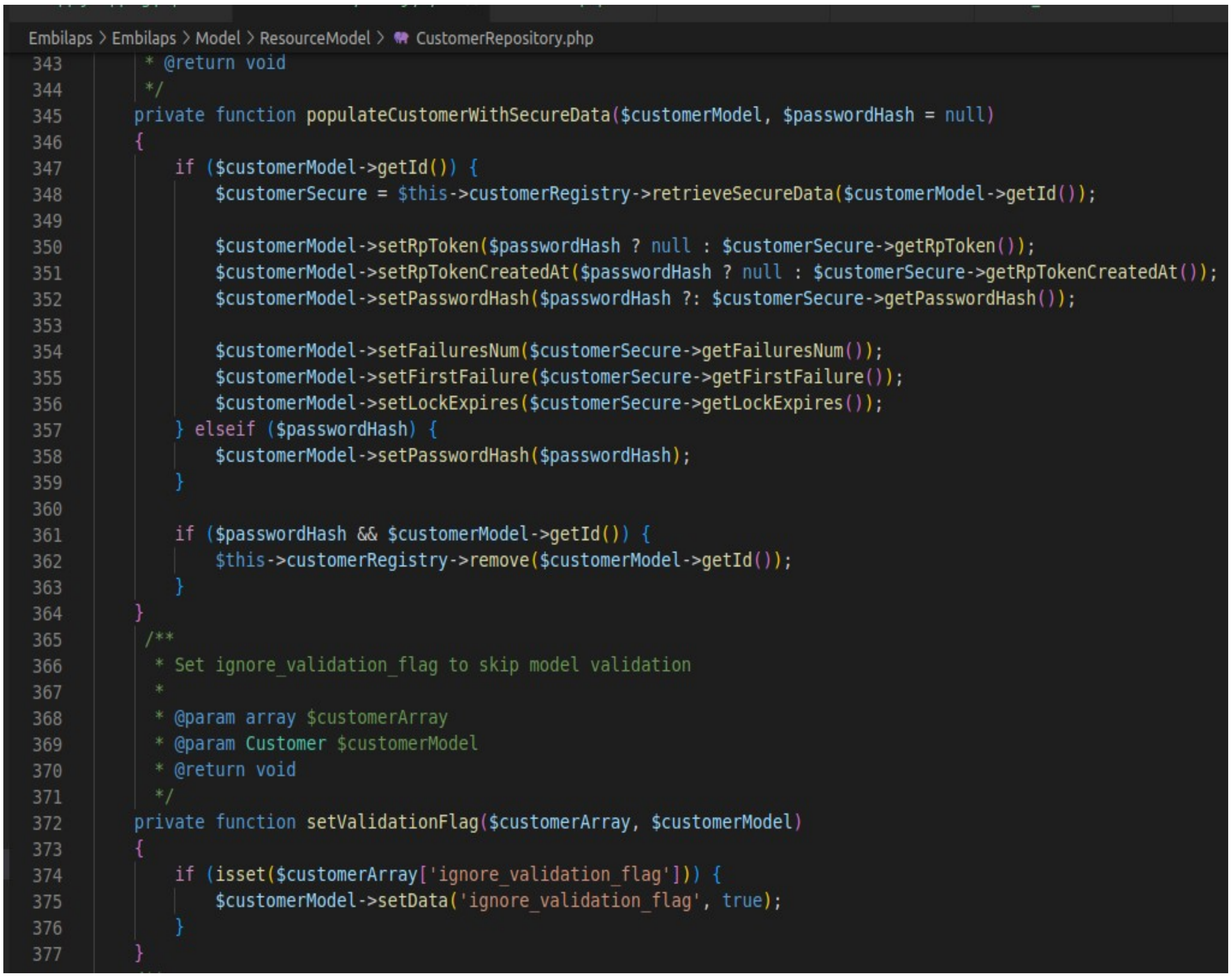
Magento/Customer/Model/Data/Customer.php

Embilaps/Embilaps/Model/ResourceModel/CustomerRepository.php



Embilaps > Embilaps > Model > ResourceModel > CustomerRepository.php

```php
<?php

namespace Embilaps\Embilaps\Model\ResourceModel;

use Magento\Customer\Api\CustomerMetadataInterface;
use Magento\Customer\Api\CustomerRepositoryInterface;
use Magento\Customer\Api\Data\CustomerInterface;
use Magento\Customer\Api\Data\CustomerSearchResultsInterfaceFactory;
use Magento\Customer\Api\GroupRepositoryInterface;
use Magento\Customer\Model\Customer as CustomerModel;
use Magento\Customer\Model\Customer\NotificationStorage;
use Magento\Customer\Model\CustomerFactory;
use Magento\Customer\Model\CustomerRegistry;
use Magento\Customer\Model\Data\CustomerSecureFactory;
use Magento\Customer\Model\Delegation\Data\NewOperation;
use Magento\Customer\Model\Delegation\Storage as DelegatedStorage;
use Magento\Customer\Model\ResourceModel\Customer\Collection;
use Magento\Framework\Api\DataObjectHelper;
use Magento\Framework\Api\ExtensibleDataObjectConverter;
use Magento\Framework\Api\ExtensionAttribute\JoinProcessorInterface;
use Magento\Framework\Api\ImageProcessorInterface;
use Magento\Framework\Api\Search\FilterGroup;
use Magento\Framework\Api\SearchCriteria\CollectionProcessorInterface;
use Magento\Framework\Api\SearchCriteriaInterface;
use Magento\Framework\App\ObjectManager;
use Magento\Framework\Event\ManagerInterface;
use Magento\Framework\Exception\LocalizedException;
use Magento\Framework\Exception\NoSuchEntityException;
use Magento\Store\Model\StoreManagerInterface;
use Magento\Customer\Model\ResourceModel\AddressRepository;
use Magento\Customer\Model\ResourceModel\Customer;

class CustomerRepository extends \Magento\Customer\Model\ResourceModel\CustomerRepository
```


Embilaps > Embilaps > Model > ResourceModel > CustomerRepository.php

```
     * @return void
     */
    private function populateCustomerWithSecureData($customerModel, $passwordHash = null)
    {
        if ($customerModel->getId()) {
            $customerSecure = $this->customerRegistry->retrieveSecureData($customerModel->getId());

            $customerModel->setRpToken($passwordHash ? null : $customerSecure->getRpToken());
            $customerModel->setRpTokenCreatedAt($passwordHash ? null : $customerSecure->getRpTokenCreatedAt());
            $customerModel->setPasswordHash($passwordHash ?: $customerSecure->getPasswordHash());

            $customerModel->setFailuresNum($customerSecure->getFailuresNum());
            $customerModel->setFirstFailure($customerSecure->getFirstFailure());
            $customerModel->setLockExpires($customerSecure->getLockExpires());
        } elseif ($passwordHash) {
            $customerModel->setPasswordHash($passwordHash);
        }

        if ($passwordHash && $customerModel->getId()) {
            $this->customerRegistry->remove($customerModel->getId());
        }
    }
     /**
      * Set ignore_validation_flag to skip model validation
      *
      * @param array $customerArray
      * @param Customer $customerModel
      * @return void
      */
    private function setValidationFlag($customerArray, $customerModel)
    {
        if (isset($customerArray['ignore_validation_flag'])) {
            $customerModel->setData('ignore_validation_flag', true);
        }
    }
```

A CARIAD GROUP COMPANY