Technical Documentation

Document ID:5459 and 5457
Document Version:1.0
Authors:Shashank G Suvarna and Pooja Bhat K
Team-7

1. Overall Description

This document contains all the functionalities that we successfully finished. Here we are creating the Career Page with all working functionalities like the admin can add the available jobs in backend, if the vacancies are closed the admin can disable the available job in the backend this is achieved using grid and form. In frontend, the user or job applicant can view all the available jobs and view specific job requirement and responsibility and apply for that job. When the applicant applies for the job the email must be generated from applicant to the admin. This email sending is achieved through SMTP.

2. Technical Requirements

2.1 GRID and FORM

Step 1: Creating the Module: In magneto 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 ‘News’ and module name as ‘Career’.

app/code/News/Career/etc/module.xml

News > Career > etc > module.xml

```
1  <?xml version="1.0"?>
2  <config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
3  <module name="News_Career" >
4      <sequence>
5          <module name="Magento_Directory"/>
6          <module name="Magento_Config"/>
7      </sequence>
8   </module>
9  </config>
10
```

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) .

app/code/News/Career/etc/acl.xml

News > Career > etc > acl.xml

```xml
<?xml version="1.0"?>
<!--
/**
 * Copyright © 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:Acl/etc/acl.xsd">
    <acl>
        <resources>
            <resource id="Magento_Backend::admin">
                <resource id="News_Career::career" title="Career" sortOrder="0">
                    <resource id="News_Career::manage" title="All Careers" sortOrder="10">

                    </resource>
                </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

app/code/News/Career/etc/db_schema.xml

News > Career > 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="career_data" resource="default" engine="innodb" comment="career_data">

      <column xsi:type="smallint" name="id" padding="7" unsigned="false" nullable="false" identity="true" comment="ID" />
      <column xsi:type="varchar" name="job_role" nullable="false" length="20" comment="job_role" />
      <column xsi:type="varchar" name="place" nullable="false" length="20" comment="Place" />
      <column xsi:type="varchar" name="email" nullable="false" length="50" comment="Email" />
      <column xsi:type="int" name="experience" nullable="false" length="20" comment="experience" />
      <column xsi:type="text" name="job_description" nullable="false" length="20" comment="Job description" />
      <column xsi:type="text" name="job_responsibility" nullable="false" length="20" comment="Job responsibility"  />
      <column xsi:type="text" name="available" nullable="false" length="20" comment="available"  />

      <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/News/Career/etc/di.xml

News > Career > 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="career_listing_data_source"
xsi:type="string">News\Career\Model\ResourceModel\Sample\Grid\Collection</item>
            </argument>
        </arguments>
    </type>
    <type name="News\Career\Model\ResourceModel\Sample\Grid\Collection">
        <arguments>
            <argument name="mainTable" xsi:type="string">career_data</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">News\Career\Model\ResourceModel\Sample</argument>
        </arguments>
    </type>


</config>

```

Step 5: Create the routes.xml in admin html: 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/News/Career/etc/adminhtml/routes.xml

News > Career > 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="career" id="career">
            <module name="News_Career" 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 registers the module.

app/code/News/Career/registration.php

News > Career > registration.php

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

```

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

app/code/News/Career/etc/adminhtml/menu.xml

News > Career > 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="News_Career::career"
title="Career" translate="title"
module="News_Career"
sortOrder="20"
resource="News_Career::career"/>
<add id="News_Career::manage" title="All Careers" translate="title" module="News_Career" sortOrder="10"
parent="News_Career::career" action="career/sample" resource="News_Career::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/News/Career/Model/Sample.php

News > Career > Model > Sample.php

```php
<?php
namespace News\Career\Model;
use Magento\Framework\Model\AbstractModel;

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

```

Step 9: Create the Sample.php in Resource Model folder:

app/code/News/Career/Model/ResourceModel/Sample.php

News > Career > Model > ResourceModel > Sample.php

```php
<?php
namespace News\Career\Model\ResourceModel;


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

```

Step 10: Create the Collection.php in Model-Resource Model-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/News/Career/Model/ResourceModel/Sample/Collection.php

News > Career > Model > ResourceModel > Sample > Collection.php

```php
<?php
namespace News\Career\Model\ResourceModel\Sample;
use Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection;



class Collection extends AbstractCollection
{
    /**

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

    protected function _construct()
    {
        $this->_init(
            'News\Career\Model\Sample',
            'News\Career\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/News/Career/Model/ResourceModel/Sample/Grid/Collection.php

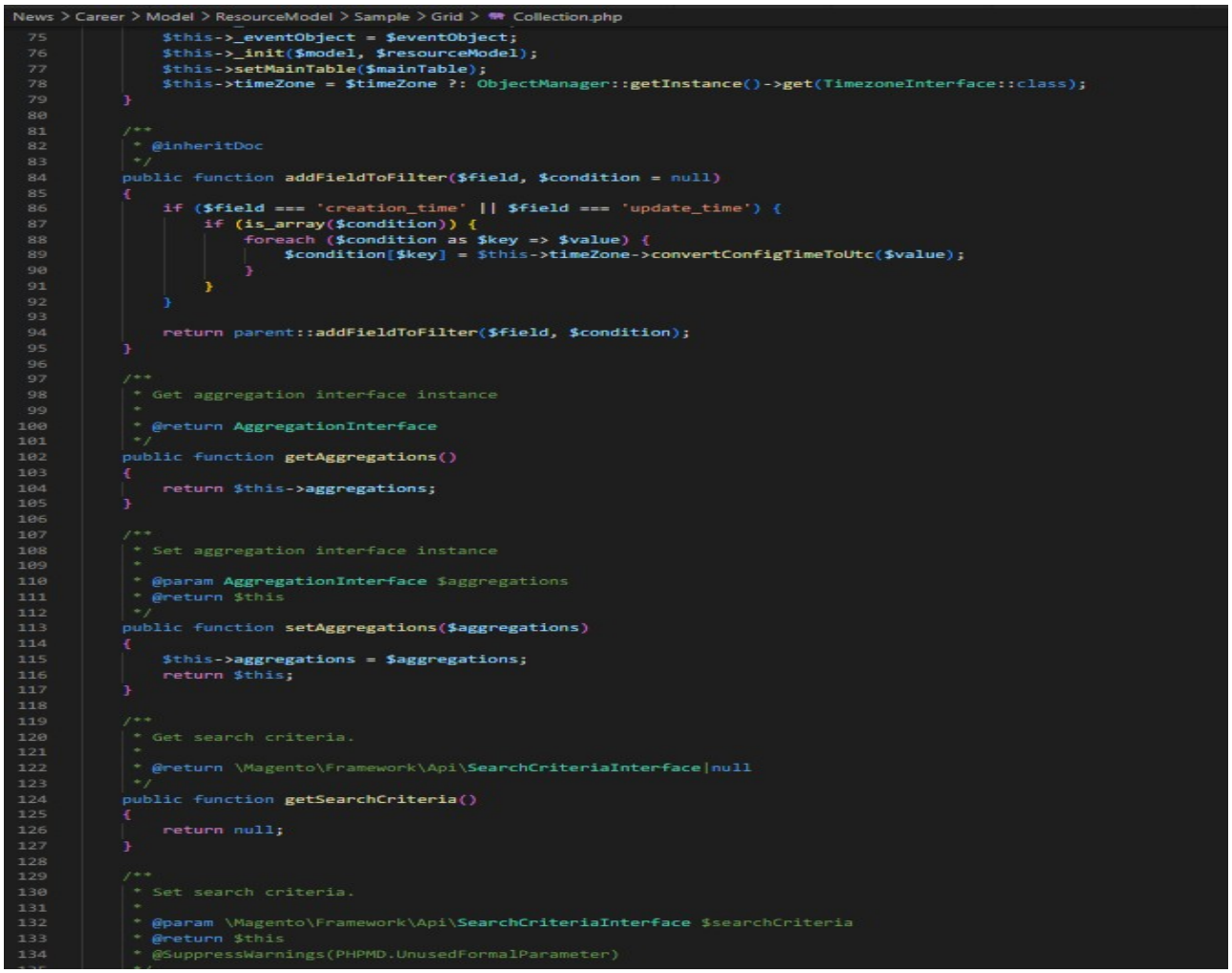
embitel
A CARIAD GROUP COMPANY

News > Career > Model > ResourceModel > Sample > Grid > Collection.php

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

use Magento\Framework\Api\Search\SearchResultInterface;
use Magento\Framework\Api\Search\AggregationInterface;
use News\Career\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 career blocks
 */
class Collection extends BlockCollection implements SearchResultInterface
{
```

News > Career > 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
        );
        $this->_eventPrefix = $eventPrefix;
        $this->_eventObject = $eventObject;
```



News > Career > Model > ResourceModel > Sample > Grid > Collection.php

```
            $this->_eventObject = $eventObject;
            $this->_init($model, $resourceModel);
            $this->setMainTable($mainTable);
            $this->timeZone = $timeZone ?: ObjectManager::getInstance()->get(TimezoneInterface::class);
        }

        /**
         * @inheritDoc
         */
        public function addFieldToFilter($field, $condition = null)
        {
            if ($field === 'creation_time' || $field === 'update_time') {
                if (is_array($condition)) {
                    foreach ($condition as $key => $value) {
                        $condition[$key] = $this->timeZone->convertConfigTimeToUtc($value);
                    }
                }
            }

            return parent::addFieldToFilter($field, $condition);
        }

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

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

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

        /**
         * Set search criteria.
         *
         * @param \Magento\Framework\Api\SearchCriteriaInterface $searchCriteria
         * @return $this
         * @SuppressWarnings(PHPMD.UnusedFormalParameter)
```

```
         */
        public function setSearchCriteria(\Magento\Framework\Api\SearchCriteriaInterface $searchCriteria = null
        {
            return $this;
        }

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

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

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

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/News/Career/Model/Grid/DataProvider.php

News > Career > Model > Grid > DataProvider.php

```php
<?php

namespace News\Career\Model\Grid;

use News\Career\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/News/Career/Controller/Adminhtml/Sample/Index.php

News > Career > Controller > Adminhtml > Sample > Index.php

```php
<?php

namespace News\Career\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='News_Career::career';
    /**
     * 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
```

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

app/code/News/Career/Controller/Adminhtml/Sample/Save.php

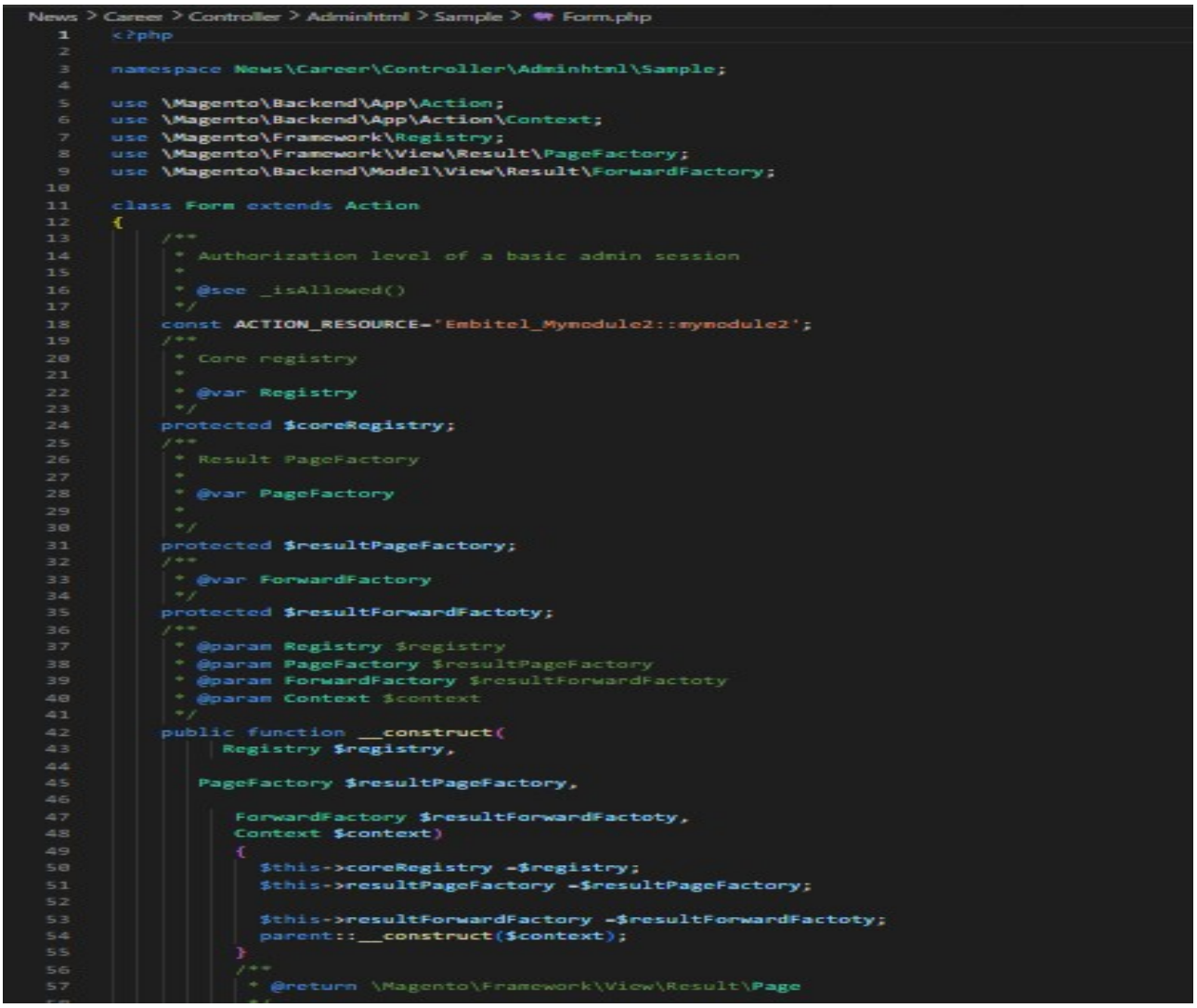
News > Career > Controller > Adminhtml > Sample > MassDelete.php

```php
<?php

namespace News\Career\Controller\Adminhtml\Sample;

use Magento\Backend\App\Action;
use Magento\Backend\App\Action\Context;
use Magento\Framework\Controller\ResultFactory;
use Magento\Ui\Component\MassAction\Filter;
use News\Career\Model\ResourceModel\Sample\CollectionFactory;

class MassDelete extends Action
{
    /**
     * Mass Action Filter
     *
     * @var Filter
     */
    protected $filter;

    /**
     * Collection Factory
     *
     * @var CollectionFactory
     */
    protected $collectionFactory;

    /**
     * Constructor
     *
     * @param Context            $context
     * @param Filter             $filter
     * @param CollectionFactory  $collectionFactory
     */
    public function __construct(
        Context $context,
        Filter $filter,
        CollectionFactory $collectionFactory
    ) {
        parent::__construct($context);
        $this->filter = $filter;
        $this->collectionFactory = $collectionFactory;
    }

    /**
     * Execute action
     *
     * @return \Magento\Backend\Model\View\Result\Redirect
     */
    public function execute()
    {
        // Get collection of samples to be deleted
        $collection = $this->filter->getCollection($this->collectionFactory->create());
        // echo($collection->getSize());die;
        // Delete each sample in the collection
        $deleteCount = 0;
        foreach ($collection->getItems() as $sample) {
            $sample->delete();
            $deleteCount++;
        }

        // Add success message
        $this->messageManager->addSuccess(__('A total of %1 record(s) have been deleted.', $deleteCount));

        // Redirect to sample grid
        $resultRedirect = $this->resultFactory->create(ResultFactory::TYPE_REDIRECT);
        $resultRedirect->setPath('*/*/index');
        return $resultRedirect;
    }

    /**
     * Check if the current user is allowed to access this action
     *
     * @return bool
     */
    protected function _isAllowed()
    {
        return $this->_authorization->isAllowed('News_Career::sample_delete');
    }
}
```

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

app/code/News/Career/Controller/Adminhtml/Sample/Form.php



Step 16: Create the career_listing.xml:

career_listing.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/News/Career/view/adminhtml/ui_component/career_listing.xml

News > Career > view > adminhtml > ui_component > career_listing.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--
/**
 * Copyright © 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">career_listing.career_listing_data_source</item>
        </item>
    </argument>
    <settings>
        <buttons>
            <button name="add">
                <url path="*/*/form"/>
                <class>primary</class>
                <label translate="true">Add New Employee</label>
            </button>
        </buttons>
        <spinner>cms_block_columns</spinner>
        <deps>
            <dep>career_listing.career_listing_data_source</dep>
        </deps>
    </settings>
    <dataSource name="career_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>News_Career::career</aclResource>
        <dataProvider class="Magento\Cms\Ui\Component\DataProvider" name="career_listing_data_source">
            <settings>
                <requestFieldName>id</requestFieldName>
                <primaryFieldName>id</primaryFieldName>
            </settings>
        </dataProvider>
    </dataSource>
    <listingToolbar name="listing_top">
        <settings>
            <sticky>true</sticky>
        </settings>
        <bookmark name="bookmarks"/>
        <columnsControls name="columns_controls"/>
        <filterSearch name="fulltext"/>
        <filters name="listing_filters">
            <settings>
                <templates>
                    <filters>
                        <select>
                            <param name="template" xsi:type="string">ui/grid/filters/elements/ui-select</param>
                            <param name="component" xsi:type="string">Magento_Ui/js/form/element/ui-select</param>
                        </select>
                    </filters>
```

```xml
                    </confirm>
                    <url path="*/*/massDelete"/>
                    <type>delete</type>
                    <label translate="true">Delete</label>
                </settings>
            </action>


        </massaction>
        <paging name="listing_paging"/>
    </listingToolbar>
    <columns name="cms_block_columns">
    <selectionsColumn name="ids">
            <settings>
                <indexField>id</indexField>
            </settings>
        </selectionsColumn>
        <settings>
            <editorConfig>
                <param name="clientConfig" xsi:type="array">
                    <item name="saveUrl" xsi:type="url" path="*/*/inlineEdit"/>
                    <item name="validateBeforeSave" xsi:type="boolean">false</item>
                </param>
                <param name="indexField" xsi:type="string">id</param>
                <param name="enabled" xsi:type="boolean">true</param>
                <param name="selectProvider" xsi:type="string">career_listing.career_listing.cms_block_columns.ids</param>
            </editorConfig>
            <childDefaults>
                <param name="fieldAction" xsi:type="array">
                    <item name="provider" xsi:type="string">career_listing.career_listing.cms_block_columns_editor</item>
                    <item name="target" xsi:type="string">startEdit</item>
                    <item name="params" xsi:type="array">
                        <item name="0" xsi:type="string">${ $.$data.rowIndex }</item>
                        <item name="1" xsi:type="boolean">true</item>
                    </item>
                </param>
            </childDefaults>
        </settings>
        <selectionsColumn name="ids">
            <settings>
                <indexField>id</indexField>
            </settings>
        </selectionsColumn>
        <column name="id">
            <settings>
                <filter>textRange</filter>
                <label translate="true">ID</label>
                <sorting>asc</sorting>
            </settings>
        </column>
        <column name="job_role">
            <settings>
                <filter>text</filter>
                <editor>
                    <validation>
                        <rule name="required-entry" xsi:type="boolean">true</rule>
                    </validation>
                    <editorType>text</editorType>
                </editor>
                <label translate="true">Job Role</label>
            </settings>
        </column>
        <column name="place">
            <settings>
                <filter>text</filter>
                <editor>
                    <validation>
                        <rule name="required-entry" xsi:type="boolean">true</rule>
                        <rule name="no-marginal-whitespace" xsi:type="boolean">true</rule>
                    </validation>
                    <editorType>text</editorType>
                </editor>
                <label translate="true">Place</label>
            </settings>
        </column>
```

Step 17: Create the career_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/News/

```
        <field name="enable">

            <argument name="data" xsi:type="array">
<item name="options" xsi:type="object">News\Career\Model\Page\Source\EnablingPin</item>
<item name="config" xsi:type="array">
<item name="label" xsi:type="string" translate="true">available</item>
<item name="dataType" xsi:type="string">text</item>
<item name="formElement" xsi:type="string">select</item>
<item name="dataScope" xsi:type="string">available</item>
</item>
</argument><settings>
<validation>
<rule name="required-entry" xsi:type="boolean">true</rule>
</validation>
</settings>
</field>

        <field name="job_role">
            <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">Job Role</item>
                    <item name="formElement" xsi:type="string">input</item>
                    <item name="source" xsi:type="string"> job_role</item>
                    <item name="dataScope" xsi:type="string">job_role </item>
                    <item name="validation" xsi:type="array">

                        <item name="required-entry" xsi:type="boolean">true</item>
                    </item>
                </item>
            </argument>
        </field>
        <field name="place">
            <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">Place</item>
                    <item name="formElement" xsi:type="string">input</item>
                    <item name="source" xsi:type="string"> place</item>
                    <item name="dataScope" xsi:type="string"> place</item>
                    <item name="validation" xsi:type="array">

                        <item name="required-entry" xsi:type="boolean">true</item>
                    </item>
                </item>
            </argument>
        </field>
        <field name="email">
            <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">Email</item>
                    <item name="formElement" xsi:type="string">email</item>
                    <item name="source" xsi:type="string">email</item>
                    <item name="dataScope" xsi:type="string">email</item>
                    <item name="validation" xsi:type="array">
```


```
        <field name="enable">

            <argument name="data" xsi:type="array">
<item name="options" xsi:type="object">News\Career\Model\Page\Source\EnablingPin</item>
<item name="config" xsi:type="array">
<item name="label" xsi:type="string" translate="true">available</item>
<item name="dataType" xsi:type="string">text</item>
<item name="formElement" xsi:type="string">select</item>
<item name="dataScope" xsi:type="string">available</item>
</item>
</argument><settings>
<validation>
<rule name="required-entry" xsi:type="boolean">true</rule>
</validation>
</settings>
</field>

        <field name="job_role">
            <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">Job Role</item>
                    <item name="formElement" xsi:type="string">input</item>
                    <item name="source" xsi:type="string"> job_role</item>
                    <item name="dataScope" xsi:type="string">job_role </item>
                    <item name="validation" xsi:type="array">

                        <item name="required-entry" xsi:type="boolean">true</item>
                    </item>
                </item>
            </argument>
        </field>
        <field name="place">
            <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">Place</item>
                    <item name="formElement" xsi:type="string">input</item>
                    <item name="source" xsi:type="string"> place</item>
                    <item name="dataScope" xsi:type="string"> place</item>
                    <item name="validation" xsi:type="array">

                        <item name="required-entry" xsi:type="boolean">true</item>
                    </item>
                </item>
            </argument>
        </field>
        <field name="email">
            <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">Email</item>
                    <item name="formElement" xsi:type="string">email</item>
                    <item name="source" xsi:type="string">email</item>
                    <item name="dataScope" xsi:type="string">email</item>
                    <item name="validation" xsi:type="array">
```

Step 19: Create the career_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/News/Career/view/adminhtml/layout/career_sample_index.xml

News > Career > view > adminhtml > layout > career_sample_index.xml

```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="career_listing"/>
        </referenceContainer>
    </body>
</page>

```

Step 20: Create the career_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 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/News/Career/view/adminhtml/layout/career_sample_form.xml

News > Career > view > adminhtml > layout > career_sample_form.xml

```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.xsd">
    <update handle="styles"/>
    <body>
        <referenceContainer name="content">
            <uiComponent name="career_form"/>
        </referenceContainer>
    </body>
</page>

```

2.2 Adding Buttons to the Form:

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

app/code/News/Career/Block/Adminhtml/Button/Back.php

News > Career > Block > Adminhtml > Button > Back.php

```php
<?php

namespace News\Career\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/News/Career/Block/Adminhtml/Button/Delete.php

News > Career > Block > Adminhtml > Button > Delete.php

```php
<?php

namespace News\Career\Block\Adminhtml\Button;

//use Magento\Backend\Block\Widget\Context;
use Magento\Framework\View\Element\UiComponent\Control\ButtonProviderInterface;

class Delete extends Generic implements ButtonProviderInterface
{
    // protected $context;

    // public function __construct(
    //     Context $context
    // ) {
    //     $this->context = $context;
    // }

    // /**
    //  * @return array
    //  */

       /**
       * @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
     */
```

3.Generic.php:

app/code/News/Career/Block/Adminhtml/Button/Generic.php

News > Career > Block > Adminhtml > Button > Generic.php

```php
<?php

namespace News\Career\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/News/Career/Block/Adminhtml/Button/Reset.php

News > Career > Block > Adminhtml > Button > Reset.php

```php
<?php

namespace News\Career\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/News/Career/Block/Adminhtml/Button/Save.php

News > Career > Block > Adminhtml > Button > Save.php

```php
<?php

namespace News\Career\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' => 'career_form.career_form',
                                'actionName' => 'save',
                                'params' => [
                                    false,
                                ],
                            ],
                        ],
                    ],
                ],
            ],
            'class_name' => Container::SPLIT_BUTTON,
            'options' => $this->getOptions(),
        ];
    }
    protected function getOptions()
    {
        $options[] = [
            'id_hard' => 'save_and_new',
            'label' => __('Save & New'),
            'data_attribute' => [
                'mage-init' => [
                    'buttonAdapter' => [
                        'actions' => [
                            [
                                'targetName' => 'career_form.career_form',
                                'actionName' => 'save',
                                'params' => [
                                    true,
                                    [
                                        'back' => 'add',
                                    ],
                                ],
                            ],
                        ],
                    ],
                ],
            ],
        ];
        $options[] = [
            'id_hard' => 'save_and_close',
            'label' => __('Save & Close'),
            'data_attribute' => [
                'mage-init' => [
                    'buttonAdapter' => [
                        'actions' => [
                            [
                                'targetName' => 'career_form.career_form',
                                'actionName' => 'save',
                                'params' => [
                                    true,
                                    [
```

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/News/Career/Controller/Adminhtml/Sample/Delete.php

News > Career > Controller > Adminhtml > Sample > Delete.php

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

use Magento\Framework\App\Action\HttpPostActionInterface;

/**
 * Delete Career page action.
 */
class Delete extends \Magento\Backend\App\Action implements HttpPostActionInterface
{
    /**
     * Authorization level of a basic admin session
     *
     * @see _isAllowed()
     */
    const ADMIN_RESOURCE = 'News_Career::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(\News\Career\Model\Sample::class);
                $model->load($id);

                $title = $model->getTitle();
                $model->delete();

                // display success message
                $this->messageManager->addSuccessMessage(__('The page has been deleted.'));

                // go to grid
                $this->_eventManager->dispatch('adminhtml_sample_on_delete', [
                    'title' => $title,
                    'status' => 'success'
                ]);

                return $resultRedirect->setPath('*/*/');
            } catch (\Exception $e) {
                $this->_eventManager->dispatch(
                    'adminhtml_sample_on_delete',
                    ['title' => $title, 'status' => 'fail']
                );
                // display error message
                $this->messageManager->addErrorMessage($e->getMessage());
                // go back to edit form
                return $resultRedirect->setPath('*/*/edit', ['id' => $id]);
            }
        }

        // display error message
        $this->messageManager->addErrorMessage(__('We can\'t find a page to delete.'));

        // go to grid
```

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/News/Career/Controller/Adminhtml/Sample/Edit.php

```php
use Magento\Backend\App\Action;

/**
 * Edit Career page action.
 */
class Edit extends \Magento\Backend\App\Action implements HttpGetActionInterface
{
    /**
     * Authorization level of a basic admin session
     *
     * @see _isAllowed()
     */
    const ADMIN_RESOURCE = 'News_Career::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
     */
    public function __construct(
        Action\Context $context,
        \Magento\Framework\View\Result\PageFactory $resultPageFactory,
        \Magento\Framework\Registry $registry
    ) {
        $this->resultPageFactory = $resultPageFactory;
        $this->_coreRegistry = $registry;
        parent::__construct($context);
    }

    /**
     * Init actions
     *
     * @return \Magento\Backend\Model\View\Result\Page
     */
    protected function _initAction()
    {
        // load layout, set active menu and breadcrumbs
        /** @var \Magento\Backend\Model\View\Result\Page $resultPage */
        $resultPage = $this->resultPageFactory->create();
        $resultPage->setActiveMenu('News_Career::career_page')
            ->addBreadcrumb(__("Career"), __("Career"))
            ->addBreadcrumb(__("Manage Pages"), __('Manage Pages'));
        return $resultPage;
    }

    /**
     * Edit Career page
     *
     * @return \Magento\Backend\Model\View\Result\Page|\Magento\Backend\Model\View\Result\Redirect
     * @SuppressWarnings(PHPMD.NPathComplexity)
     */
    public function execute()
    {
        // 1. Get ID and create model
        $id = $this->getRequest()->getParam('id');

        $model = $this->_objectManager->create(\News\Career\Model\Sample::class);

        // 2. Initial checking
        if ($id) {
            $model->load($id);
            if (!$model->getId()) {
                $this->messageManager->addErrorMessage(__('This page no longer exists.'));
                /** \Magento\Backend\Model\View\Result\Redirect $resultRedirect */
                $resultRedirect = $this->resultRedirectFactory->create();
                return $resultRedirect->setPath('*/*/');
            }
        }

        //$this->_coreRegistry->register('cms_page', $model);

        // 5. Build edit form
        /** @var \Magento\Backend\Model\View\Result\Page $resultPage */
        $resultPage = $this->_initAction();
        $resultPage->addBreadcrumb(
            $id ? __('Edit Page') : __('New Page'),
            $id ? __('Edit Page') : __('New Page')
        );
        $resultPage->getConfig()->getTitle()->prepend(__('Pages'));
```

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/News/Career/Ui/Component/Listing/Column/PageActions.php

News > Career > Ui > Component > Listing > Column > PageActions.php

```php
<?php
/**
 * Copyright © Magento, Inc. All rights reserved.
 * See COPYING.txt for license details.
 */
namespace News\Career\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 URL_PATH_EDIT = 'career/sample/edit';
    const URL_PATH_DELETE = 'career/sample/delete';
    /**
     * @var \Magento\Cms\Block\Adminhtml\Page\Grid\Renderer\Action\UrlBuilder
     */
    protected $actionUrlBuilder;

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

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


    /**
     * @param ContextInterface $context
     * @param UiComponentFactory $uiComponentFactory
     * @param UrlBuilder $actionUrlBuilder
     * @param UrlInterface $urlBuilder
     * @param array $components
     * @param array $data
     * @param string $editUrl
     * @param \News\Career\ViewModel\Page\Grid\UrlBuilder|null $scopeUrlBuilder
     */
    public function __construct(
        ContextInterface $context,
        UiComponentFactory $uiComponentFactory,
        UrlBuilder $actionUrlBuilder,
        UrlInterface $urlBuilder,
        array $components = [],
        array $data = [],
        $editUrl = self::URL_PATH_EDIT,
        \News\Career\ViewModel\Page\Grid\UrlBuilder $scopeUrlBuilder = null
    ) {
        $this->urlBuilder = $urlBuilder;
        $this->editUrl = $editUrl;
        parent::__construct($context, $uiComponentFactory, $components, $data);
        $this->scopeUrlBuilder = $scopeUrlBuilder ?: ObjectManager::getInstance()
            ->get(\News\Career\ViewModel\Page\Grid\UrlBuilder::class);
    }

    /**
     * @inheritDoc
     */
    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']]),
```

News > Career > Ui > Component > Listing > Column > PageActions.php

```php
                    ];


                    $id = $this->getEscaper()->escapeHtml($item['id']);
                    $item[$name]['delete'] = [
                        'href' => $this->urlBuilder->getUrl(self::URL_PATH_DELETE, ['id' => $item['id']]),
                        'label' => __('Delete'),
                        'confirm' => [
                            'id' => __('Delete %1', $id),
                            'message' => __('Are you sure you want to delete a %1 th record?', $id),
                        ],
                        'post' => true,
                    ];
                }
                if (isset($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;
    }

    /**
     * Get instance of escaper
     *
     * @return Escaper
     * @deprecated 101.0.7
     */
    private function getEscaper()
    {
        if (!$this->escaper) {
            $this->escaper = ObjectManager::getInstance()->get(Escaper::class);
        }
        return $this->escaper;
    }
}
```

3. Front-End:

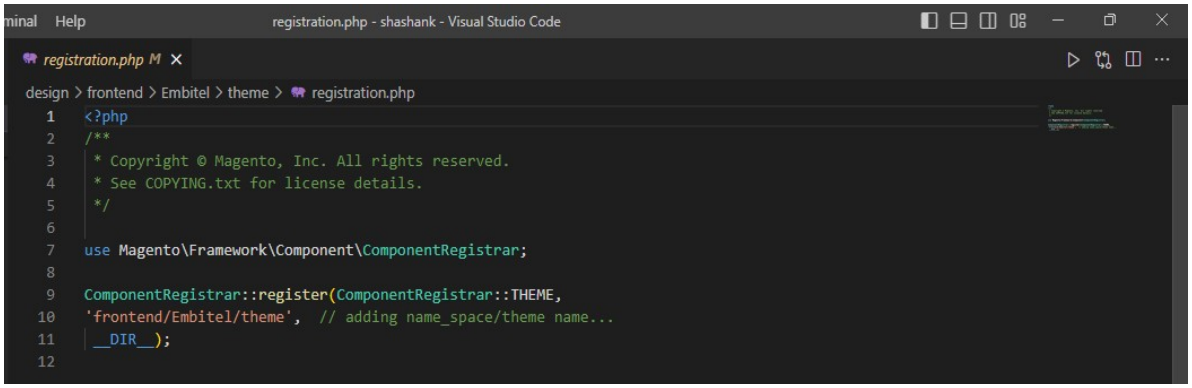
3.1 Creating Theme and Subtheme:

I utilised a custom theme and a child theme for this project, which inherit the parent theme's characteristics. The path of the theme is - app/design/frontend/Embitel/theme

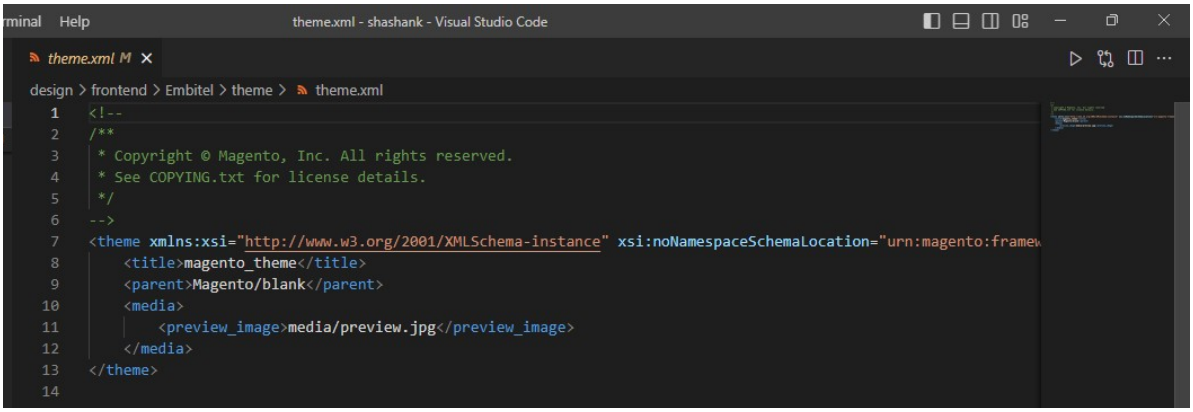
app/design/frontend/Embitel/subtheme

Below steps are followed while creating Theme:

- First I created registration.php for registering theme.

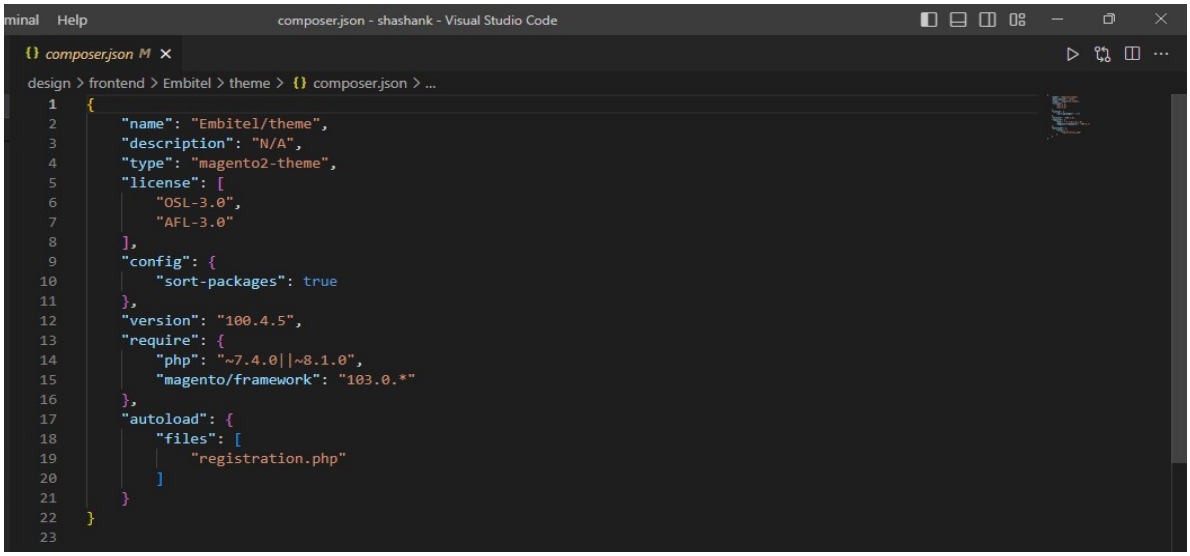


- Then created theme.xml for the theme



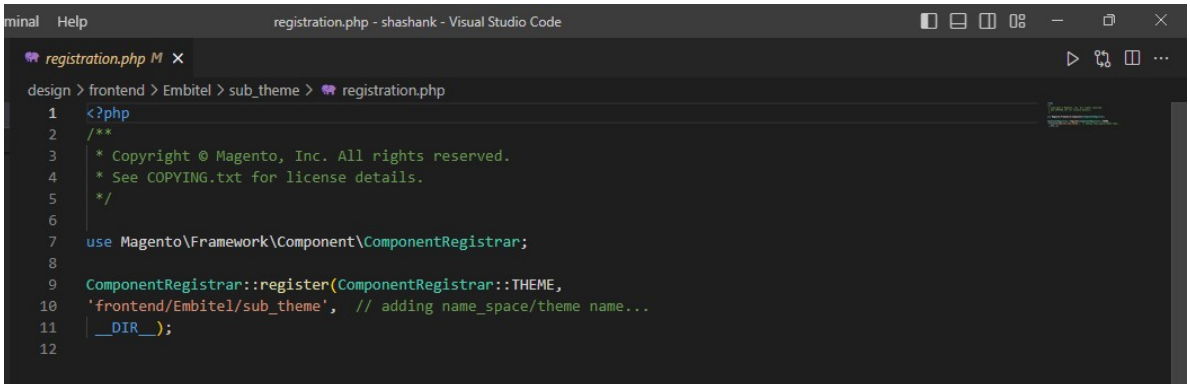
In here parent theme for the Theme is magento's Blank theme and for the Subtheme's parent theme is Embitel theme which we created.

- Then created composer.json

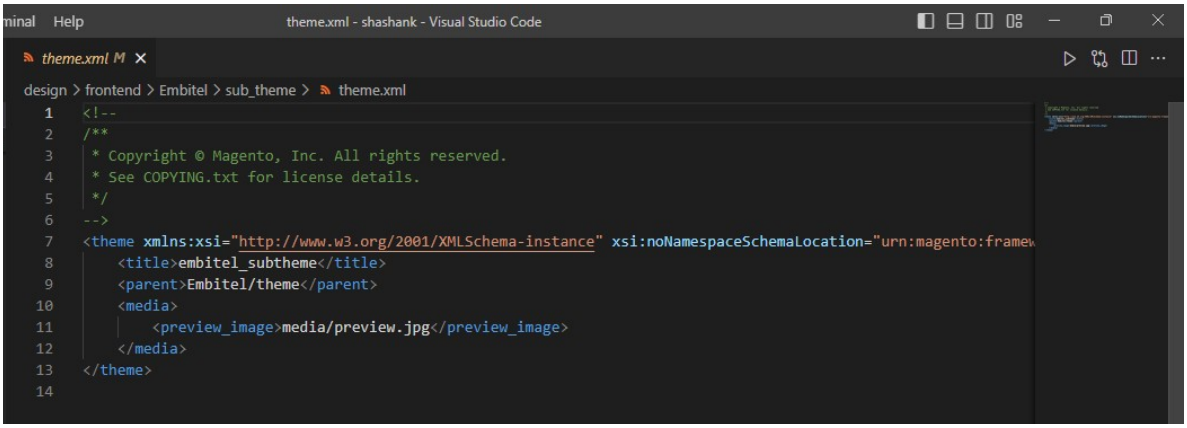


Creating a subtheme

- Creating registration.php

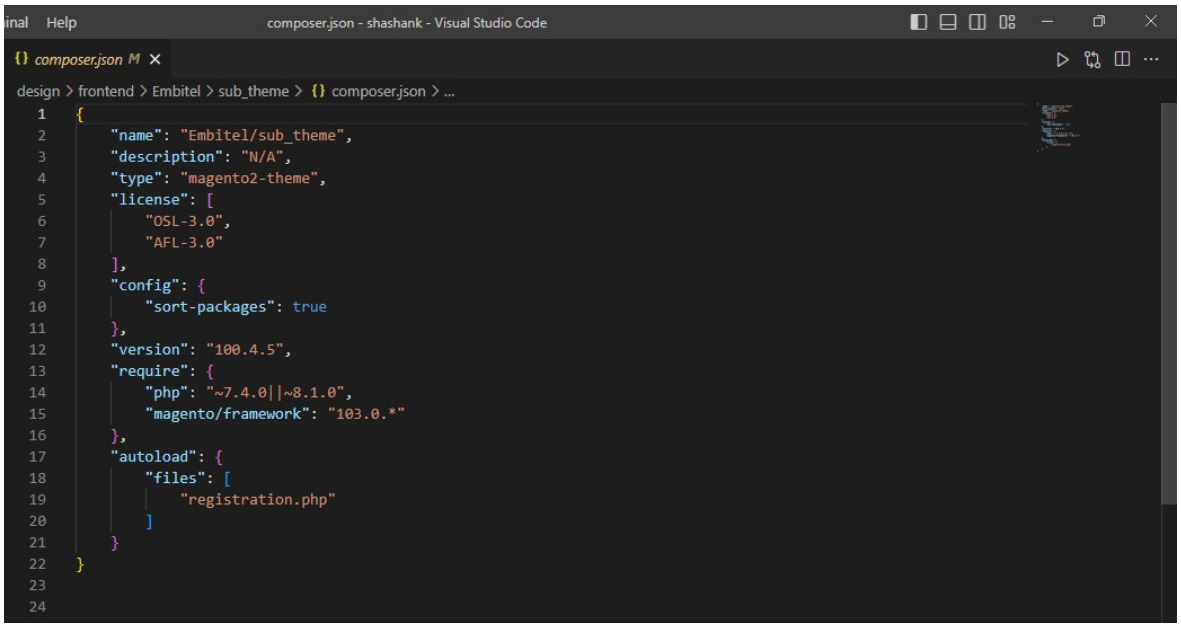


- Creating theme.xml



In here parent theme for the Theme is Embitel

- Creating composer.json

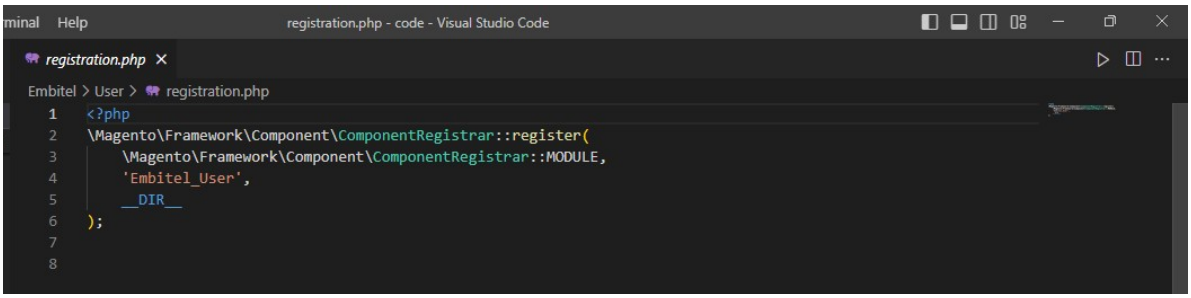


Creating Module:

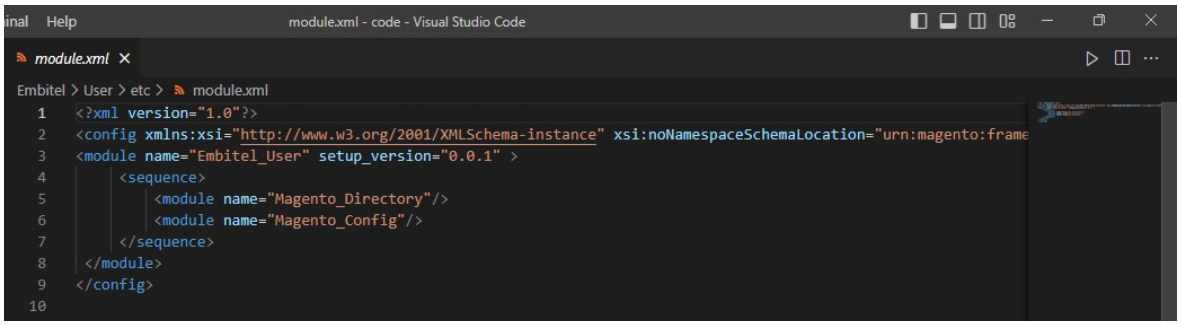
In this project we have created a module called “User” in the “Embitel” namespace.

Below steps are used for create a module –

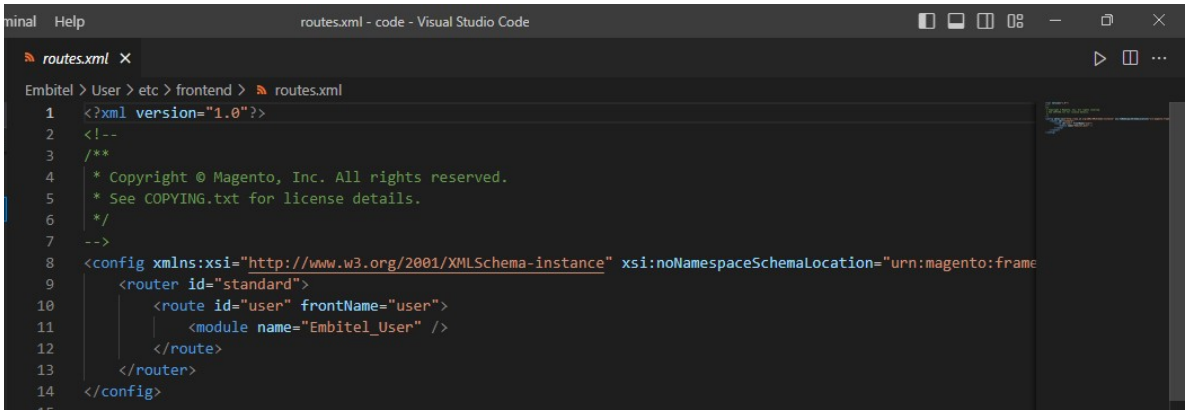
- First we created registration.php for registering the module.



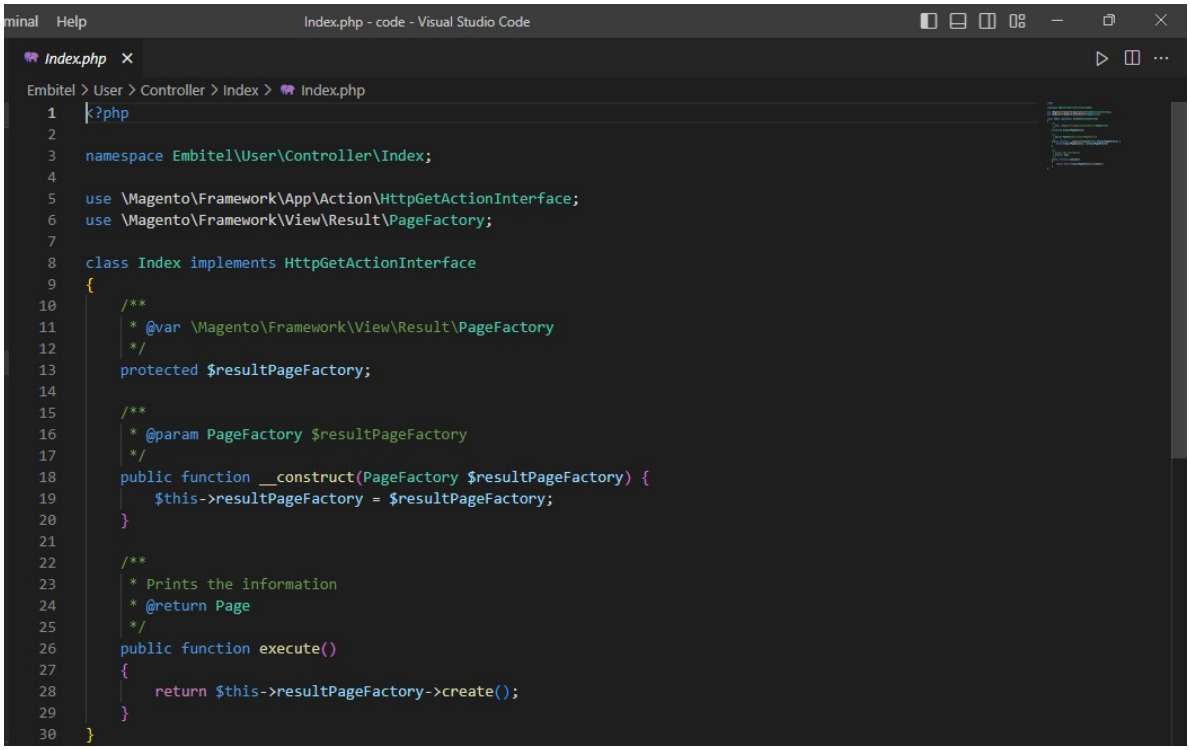
- Then created *module.xml* inside *etc* folder- etc/module.xml



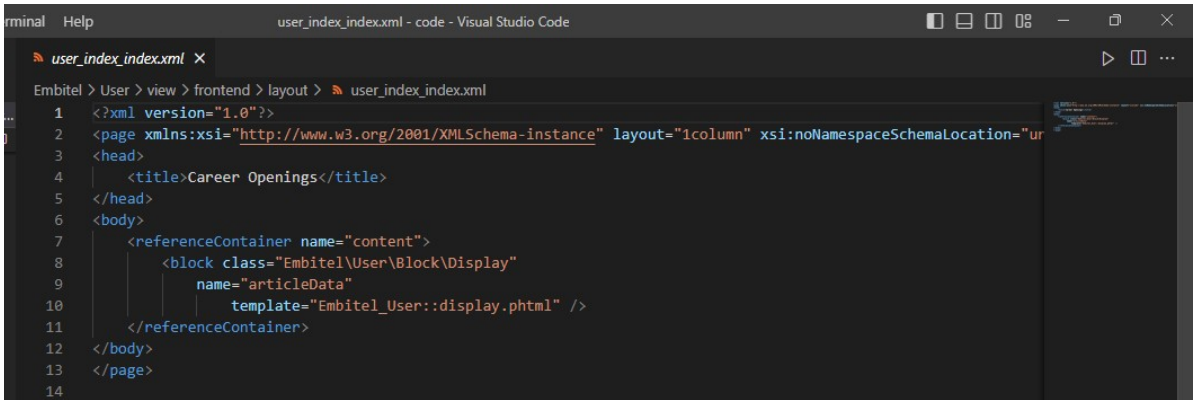
- Then we have to create the router for that in the we have to create *routes.xml* in the etc folder – etc/frontend/routes.xml



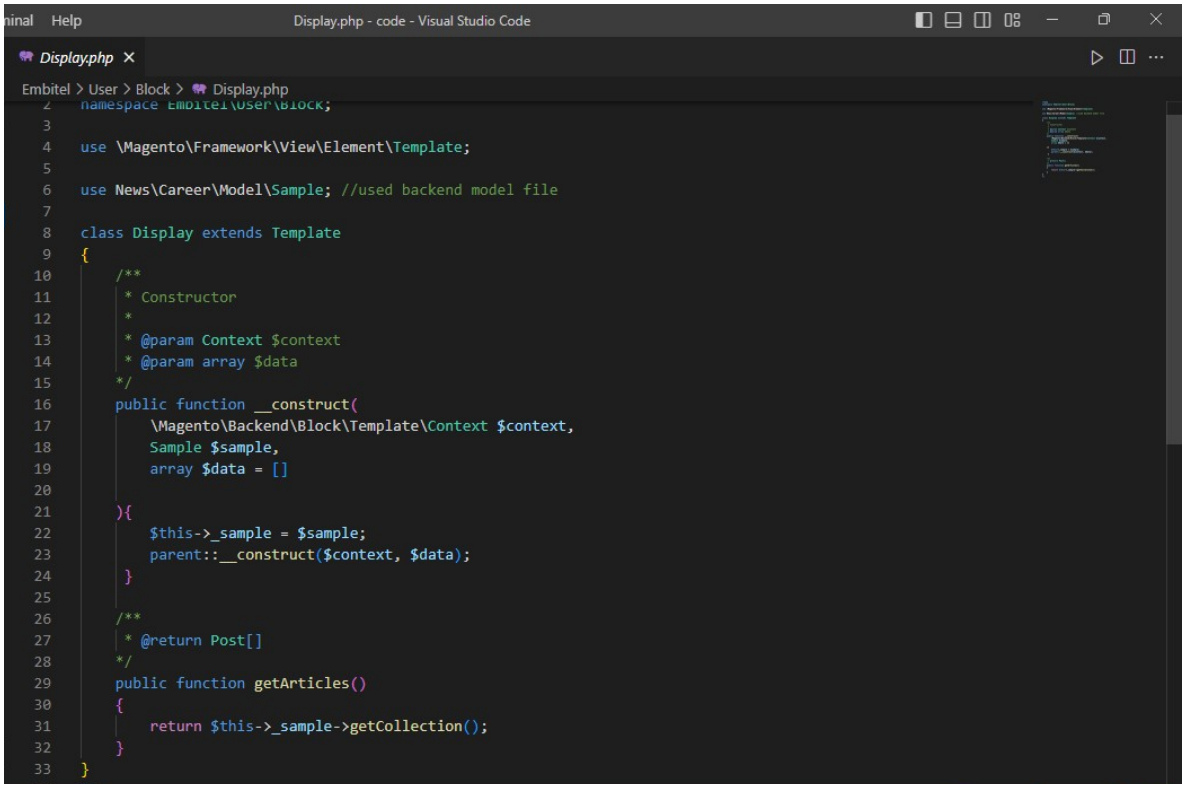
- After *routes.xml* we have to control for controlling the routes inside the *Controller* folder – Controller/index/Index.php



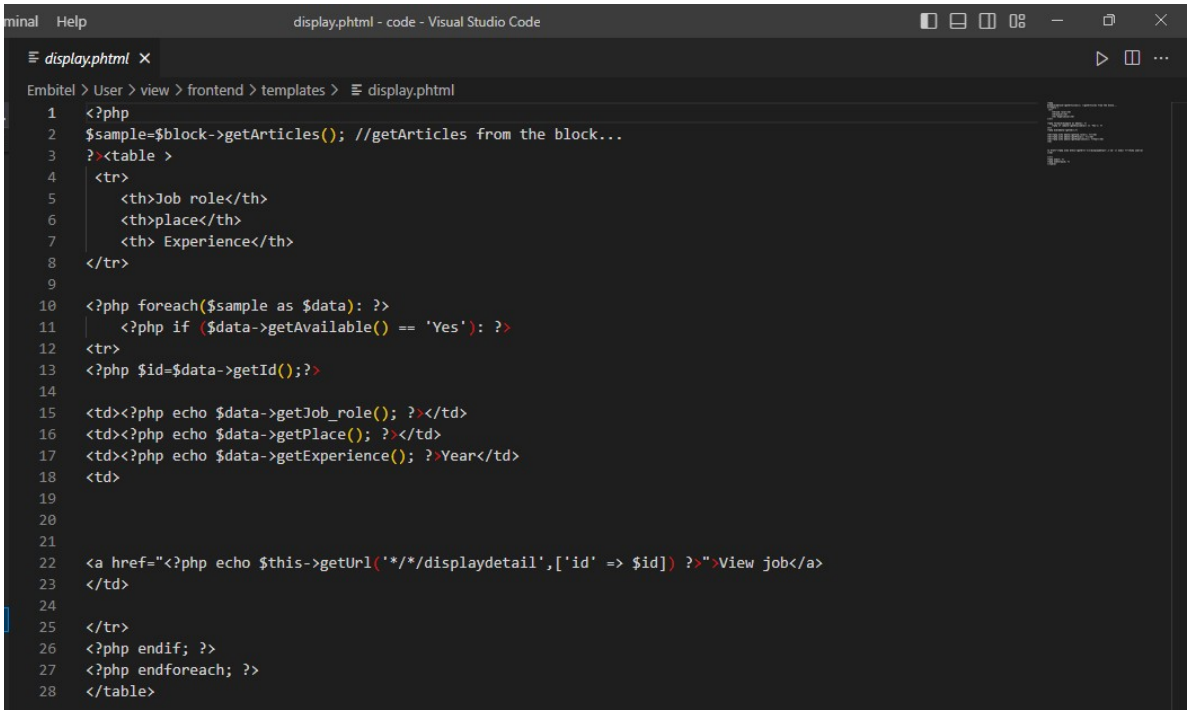
- we have to create layout file named as moduleName_folder_controllerFileName.xml, in here User is our module and inside controller index is the folder inside that index.php is the controller file. So file name is user_index_index.xml



- Here I have called block class that is Display.php which resides inside Embitel/User/Block

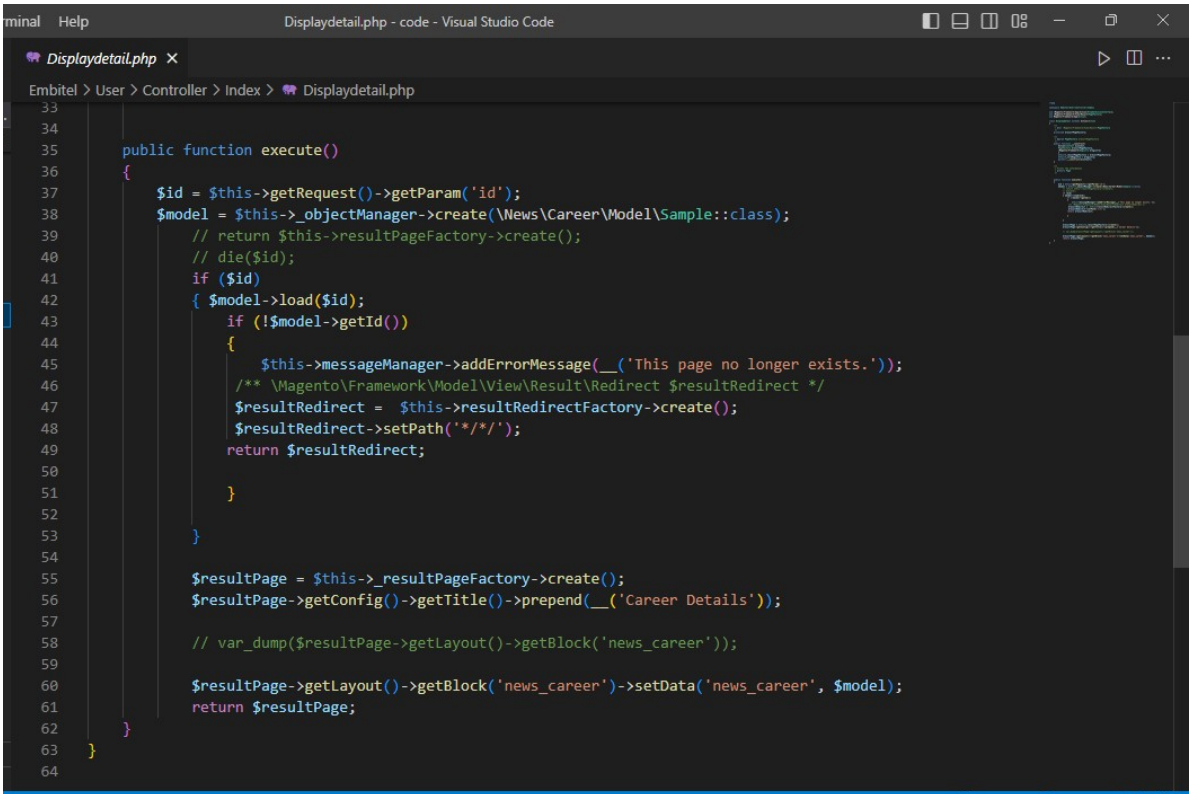


- After we have to create the *display.phtml* file inside the *view/frontend/templates/display.phtml*



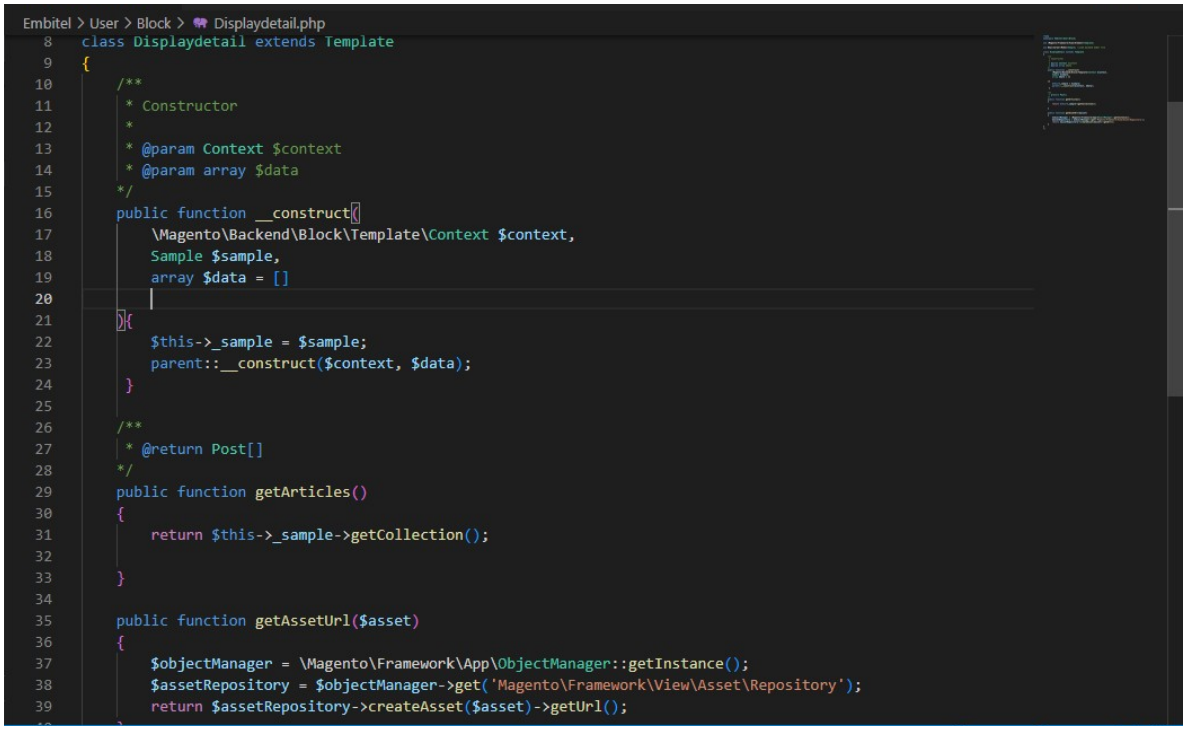
- in display.phtml file we are displaying the available jobs, if they are interested they can view and apply to the job in next page
- To rediredt to next page I have called displaydetail controller class in the above mentioned file

- Displaydetail.php controller will diplay the detailed display of the job and it provide application form in a same page



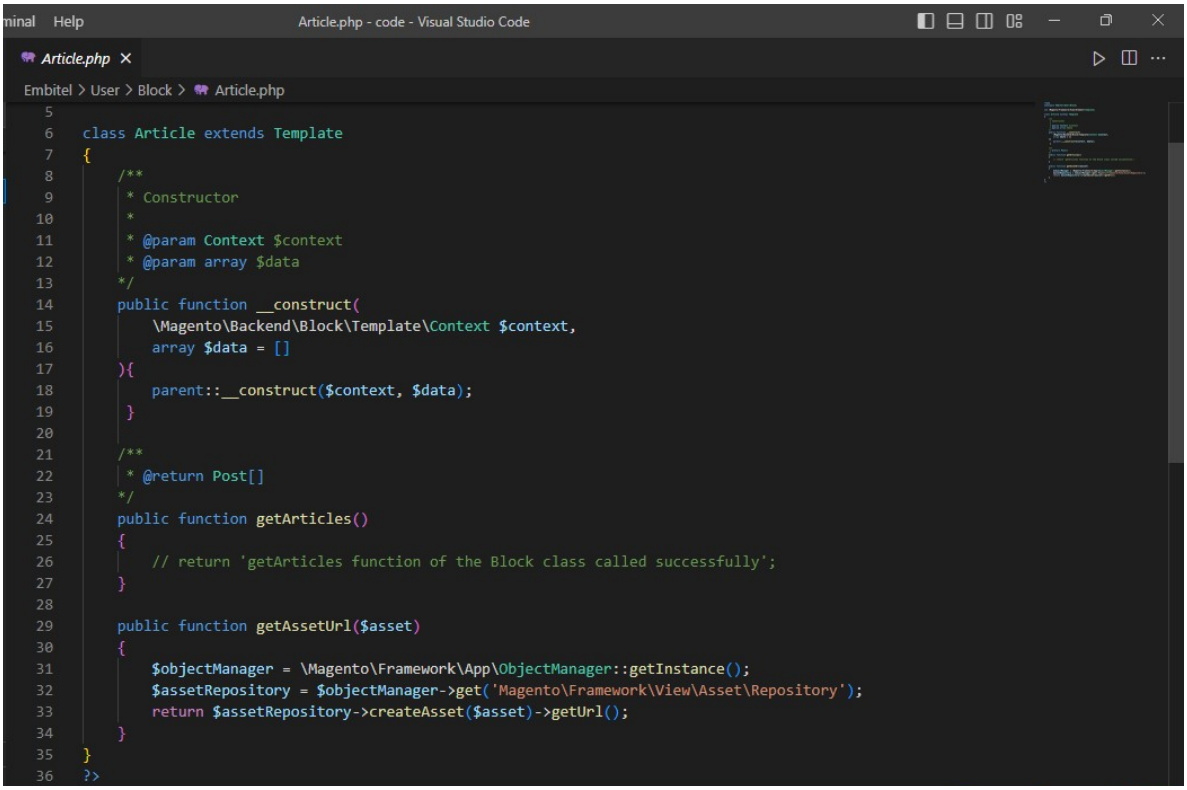
- To display application form and job details I created two block class Article.php and Displaydetail.php
- In block folder display.php and displaydetail.php I have used
- *News/Career/Model/Sample to access the model file*

- *Displaydetail.php*



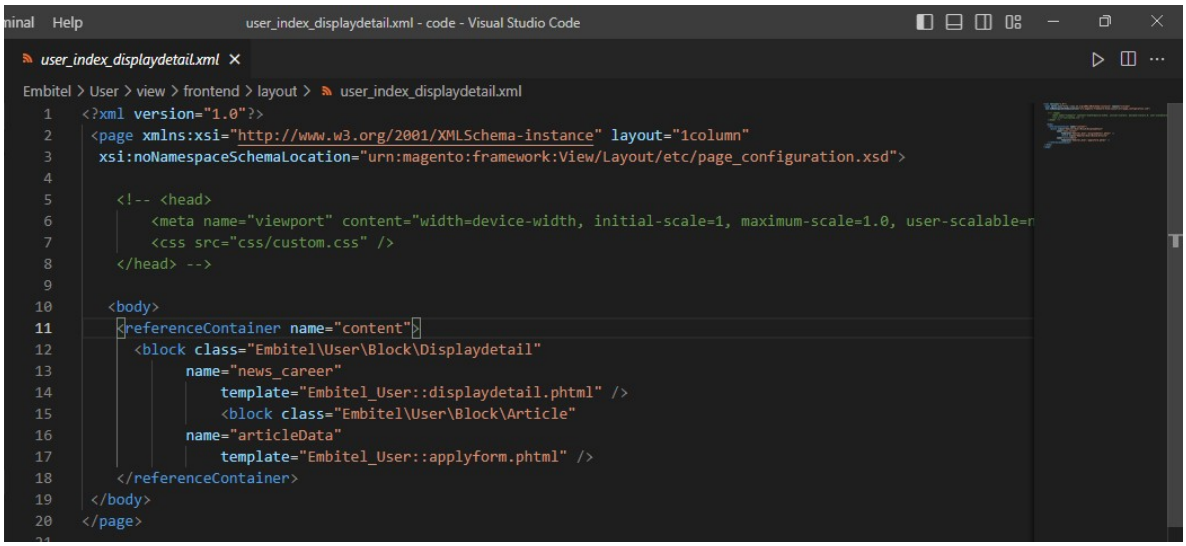

```php
class Displaydetail extends Template
{
    /**
     * Constructor
     *
     * @param Context $context
     * @param array $data
     */
    public function __construct(
        \Magento\Backend\Block\Template\Context $context,
        Sample $sample,
        array $data = []

    ){
        $this->_sample = $sample;
        parent::__construct($context, $data);
     }

    /**
     * @return Post[]
     */
    public function getArticles()
    {
        return $this->_sample->getCollection();

    }

    public function getAssetUrl($asset)
    {
        $objectManager = \Magento\Framework\App\ObjectManager::getInstance();
        $assetRepository = $objectManager->get('Magento\Framework\View\Asset\Repository');
        return $assetRepository->createAsset($asset)->getUrl();
```

- Article.php

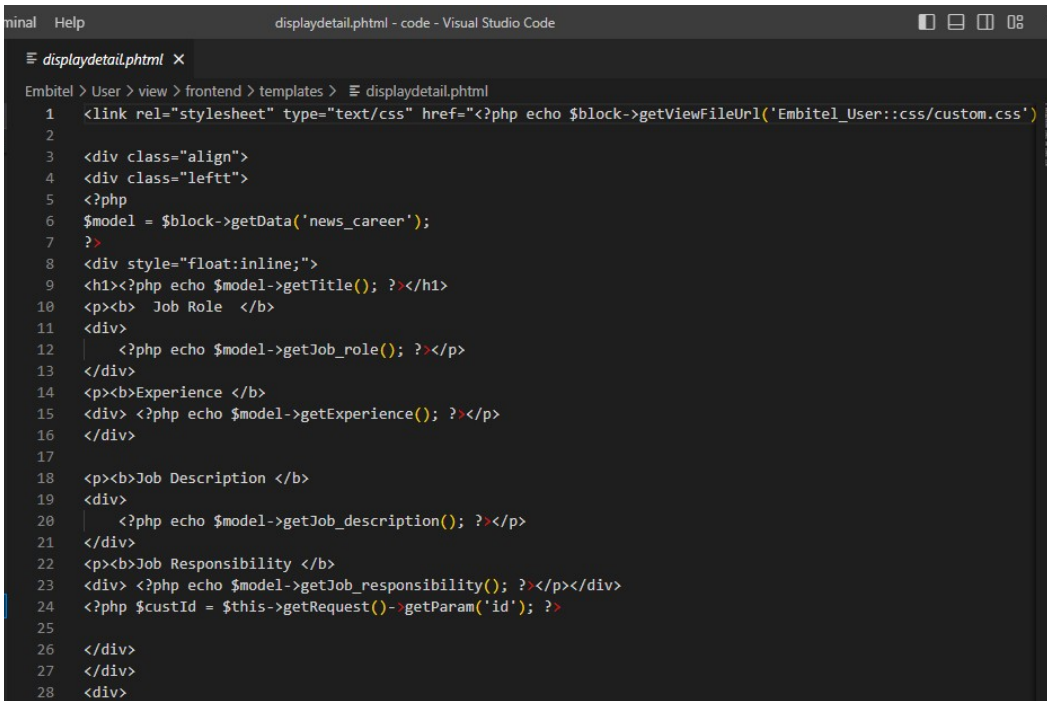



```php
class Article extends Template
{
    /**
     * Constructor
     *
     * @param Context $context
     * @param array $data
     */
    public function __construct(
        \Magento\Backend\Block\Template\Context $context,
        array $data = []
    ){
        parent::__construct($context, $data);
     }

    /**
     * @return Post[]
     */
    public function getArticles()
    {
        // return 'getArticles function of the Block class called successfully';
    }

    public function getAssetUrl($asset)
    {
        $objectManager = \Magento\Framework\App\ObjectManager::getInstance();
        $assetRepository = $objectManager->get('Magento\Framework\View\Asset\Repository');
        return $assetRepository->createAsset($asset)->getUrl();
    }
}
?>
```

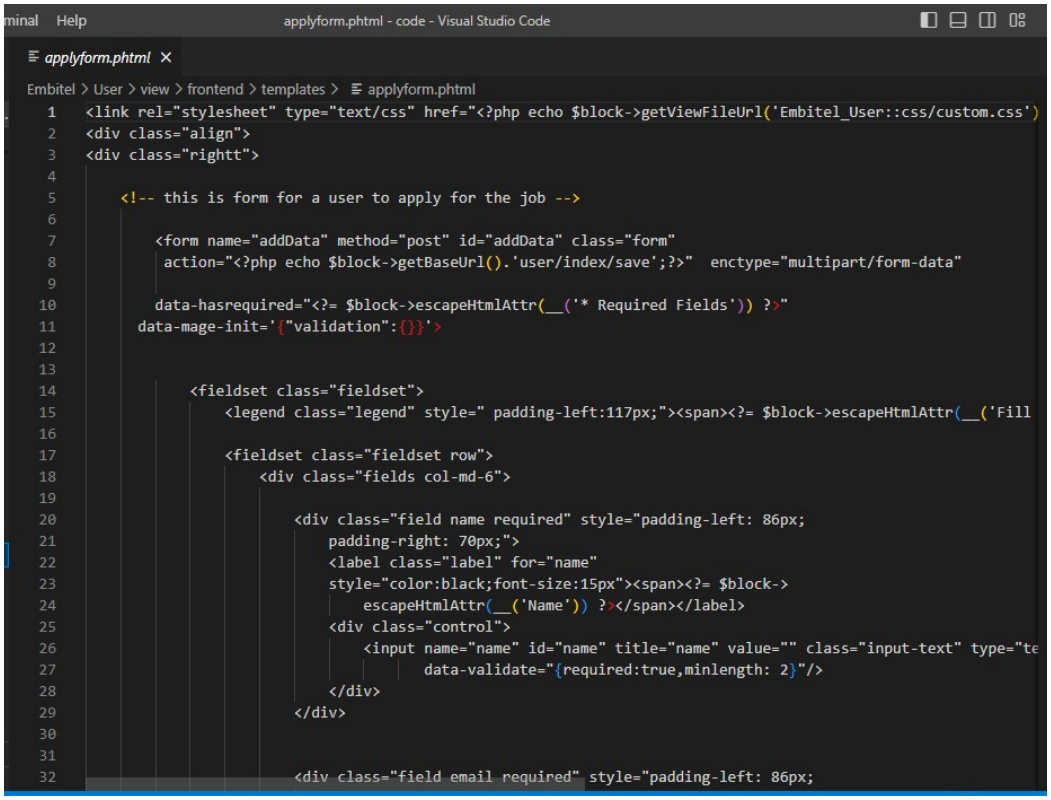
- Layout file user_index_displaydeatil.xml here I added two block class inside reference container two display job description and application form



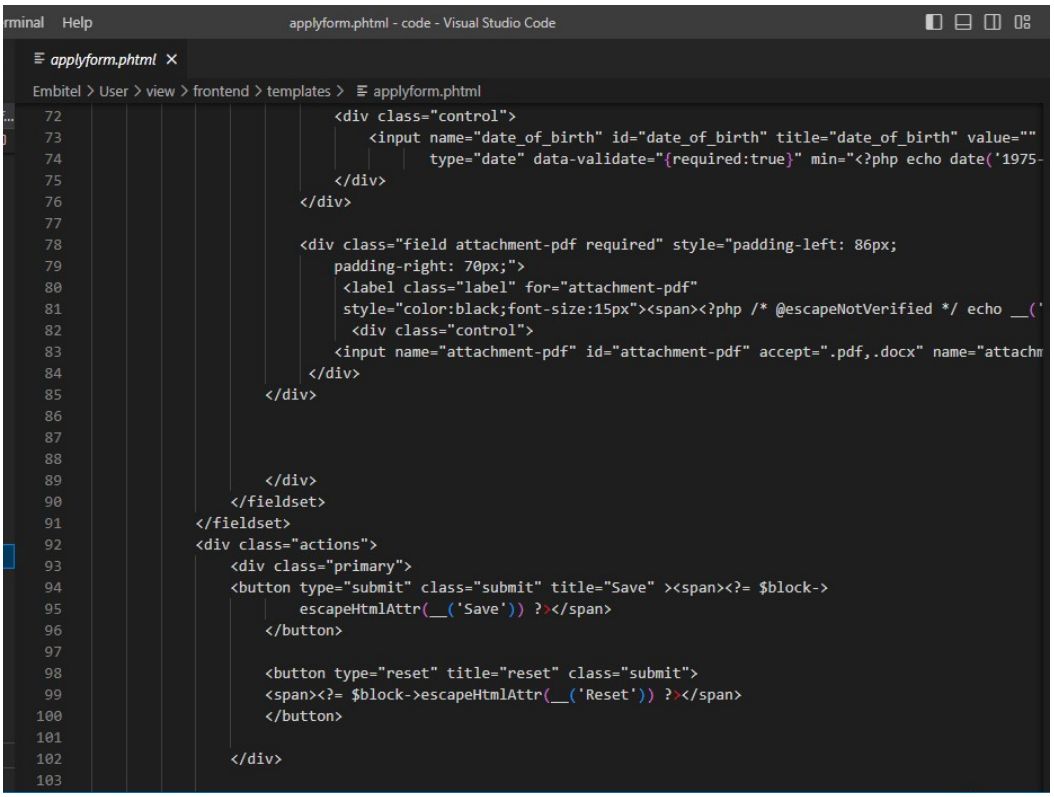
- Two templates are called in the above layout file
- Displaydetail.phtml and apply.phtml



- Apply.phtml

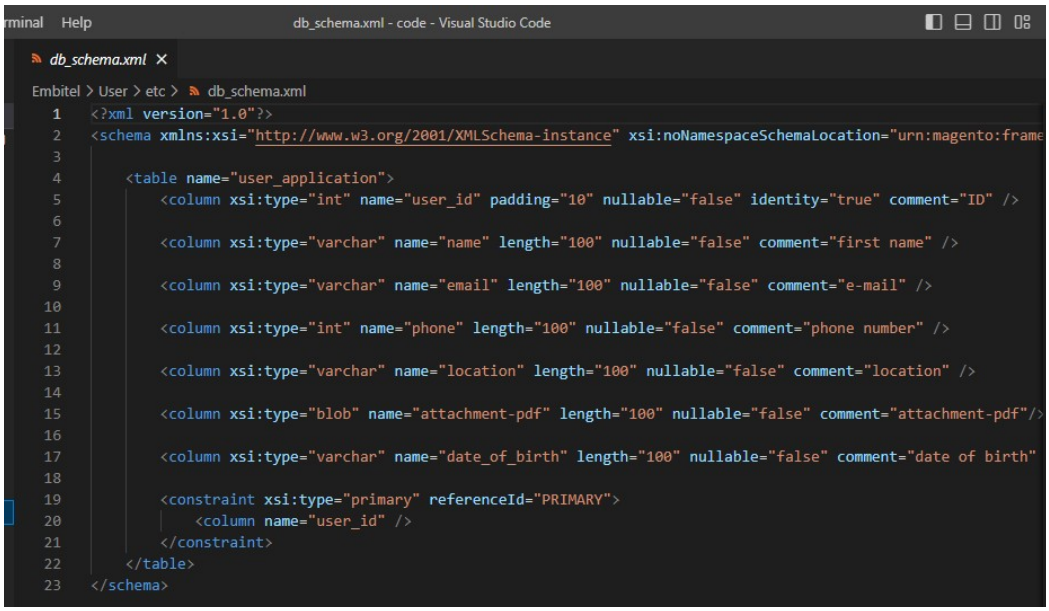



```
<link rel="stylesheet" type="text/css" href="<?php echo $block->getViewFileUrl('Embitel_User::css/custom.css')
<div class="align">
<div class="rightt">

    <!-- this is form for a user to apply for the job -->

        <form name="addData" method="post" id="addData" class="form"
         action="<?php echo $block->getBaseUrl().'user/index/save';?>"  enctype="multipart/form-data"

        data-hasrequired="<?= $block->escapeHtmlAttr(__('* Required Fields')) ?>"
       data-mage-init='{"validation":{}}'>


            <fieldset class="fieldset">
                <legend class="legend" style=" padding-left:117px;"><span><?= $block->escapeHtmlAttr(__('Fill

                <fieldset class="fieldset row">
                    <div class="fields col-md-6">

                        <div class="field name required" style="padding-left: 86px;
                            padding-right: 70px;">
                            <label class="label" for="name"
                            style="color:black;font-size:15px"><span><?= $block->
                                escapeHtmlAttr(__('Name')) ?></span></label>
                            <div class="control">
                                <input name="name" id="name" title="name" value="" class="input-text" type="te
                                        data-validate="{required:true,minlength: 2}"/>
                            </div>
                        </div>


                        <div class="field email required" style="padding-left: 86px;
```

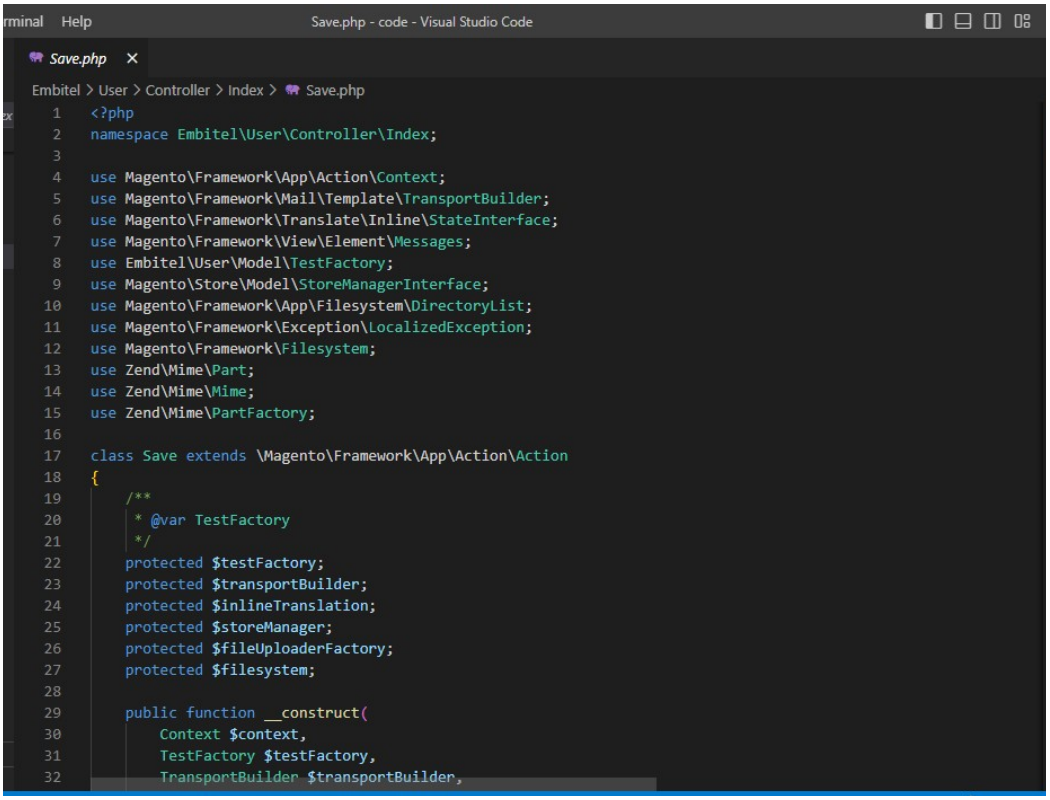


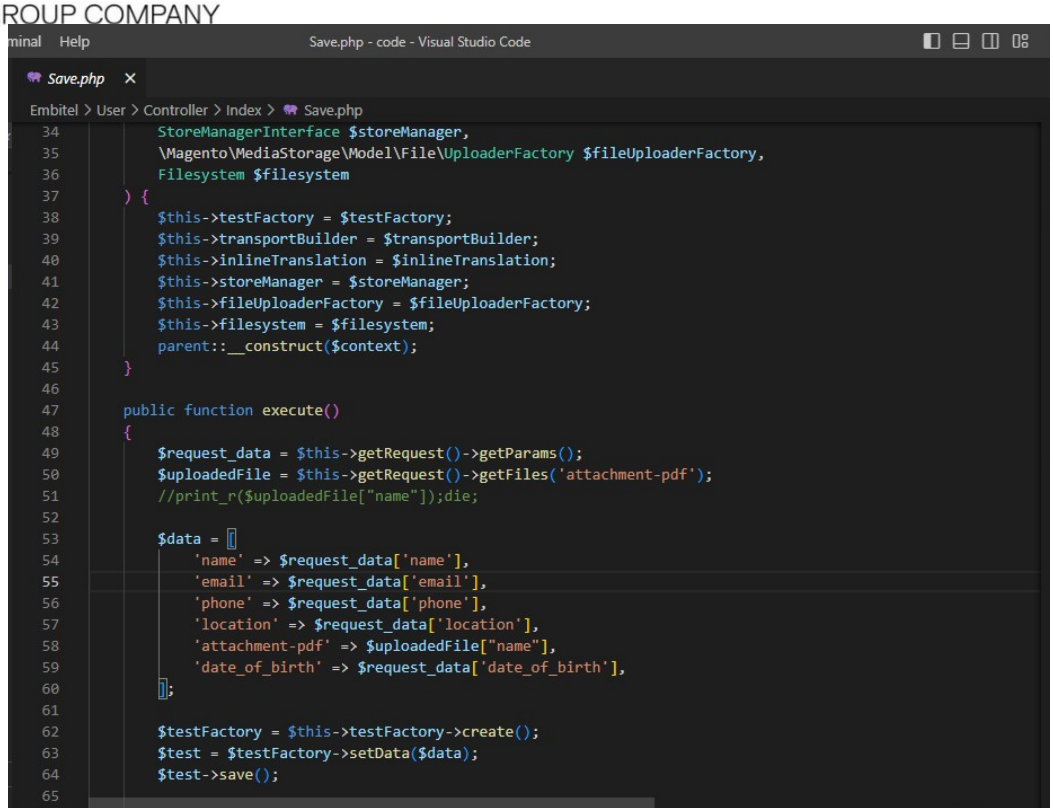

```
                            <div class="control">
                                <input name="date_of_birth" id="date_of_birth" title="date_of_birth" value=""
                                        type="date" data-validate="{required:true}" min="<?php echo date('1975-
                            </div>
                        </div>

                        <div class="field attachment-pdf required" style="padding-left: 86px;
                            padding-right: 70px;">
                             <label class="label" for="attachment-pdf"
                             style="color:black;font-size:15px"><span><?php /* @escapeNotVerified */ echo __('
                              <div class="control">
                             <input name="attachment-pdf" id="attachment-pdf" accept=".pdf,.docx" name="attachm
                            </div>
                    </div>


                </div>
            </fieldset>
        </fieldset>
        <div class="actions">
            <div class="primary">
            <button type="submit" class="submit" title="Save" ><span><?= $block->
                    escapeHtmlAttr(__('Save')) ?></span>
                </button>

                <button type="reset" title="reset" class="submit">
                <span><?= $block->escapeHtmlAttr(__('Reset')) ?></span>
                </button>

            </div>
```

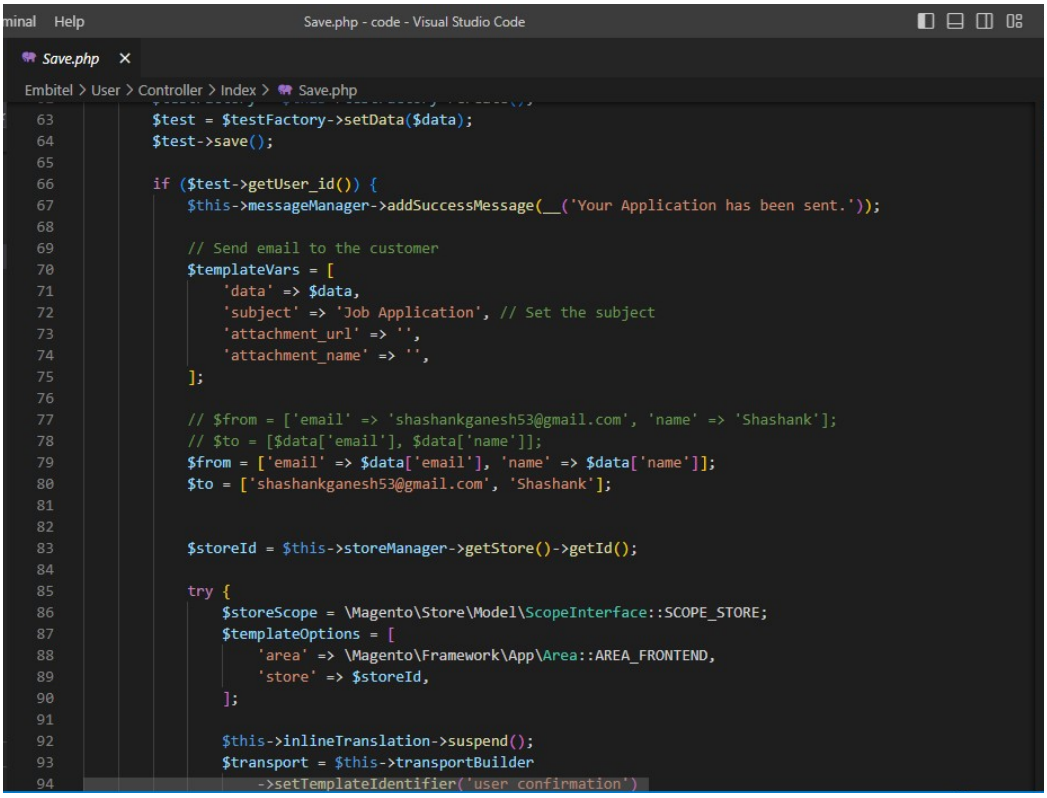
- When user enters tha form field it should be stored in DB so I created db_schema inside etc folder

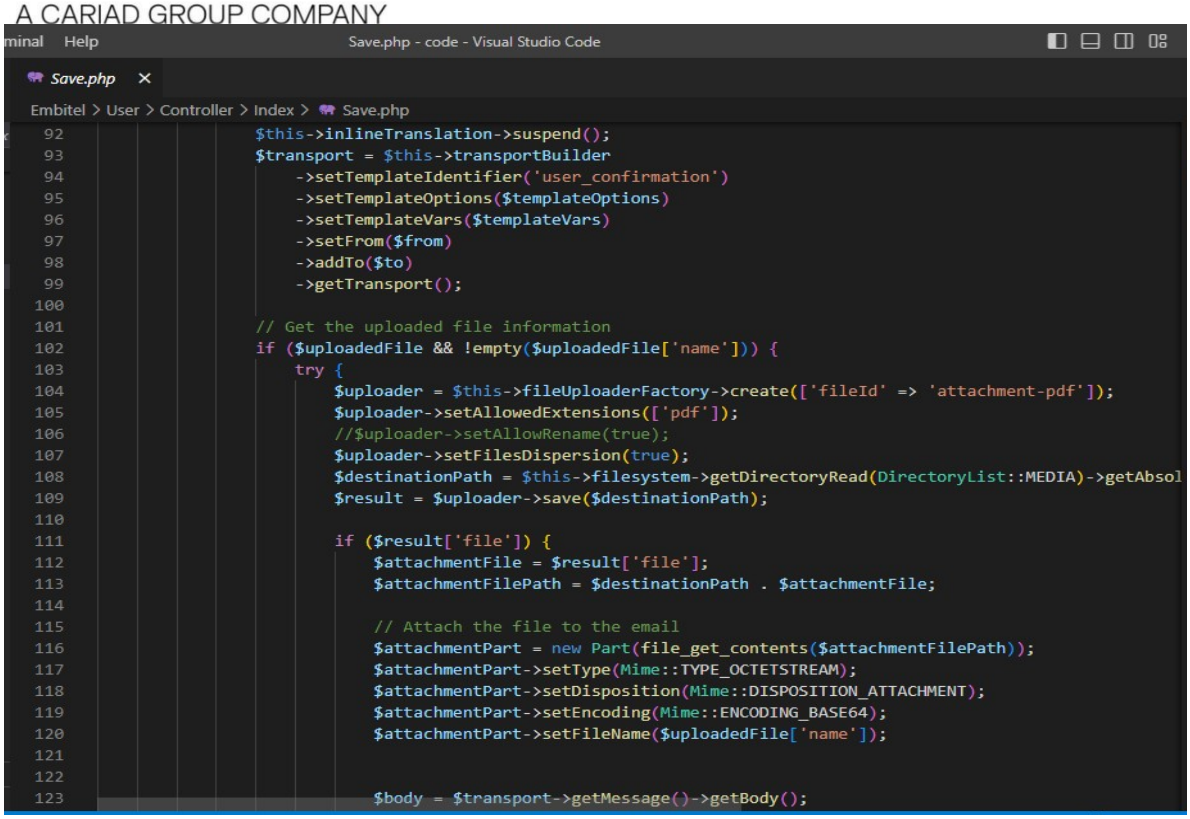


- To store the data in database I created Save.php in controller and here I am sending mail from applicant to admin.
- In mail attaching the resume also, so in save.php we have writen logic for sending mail also

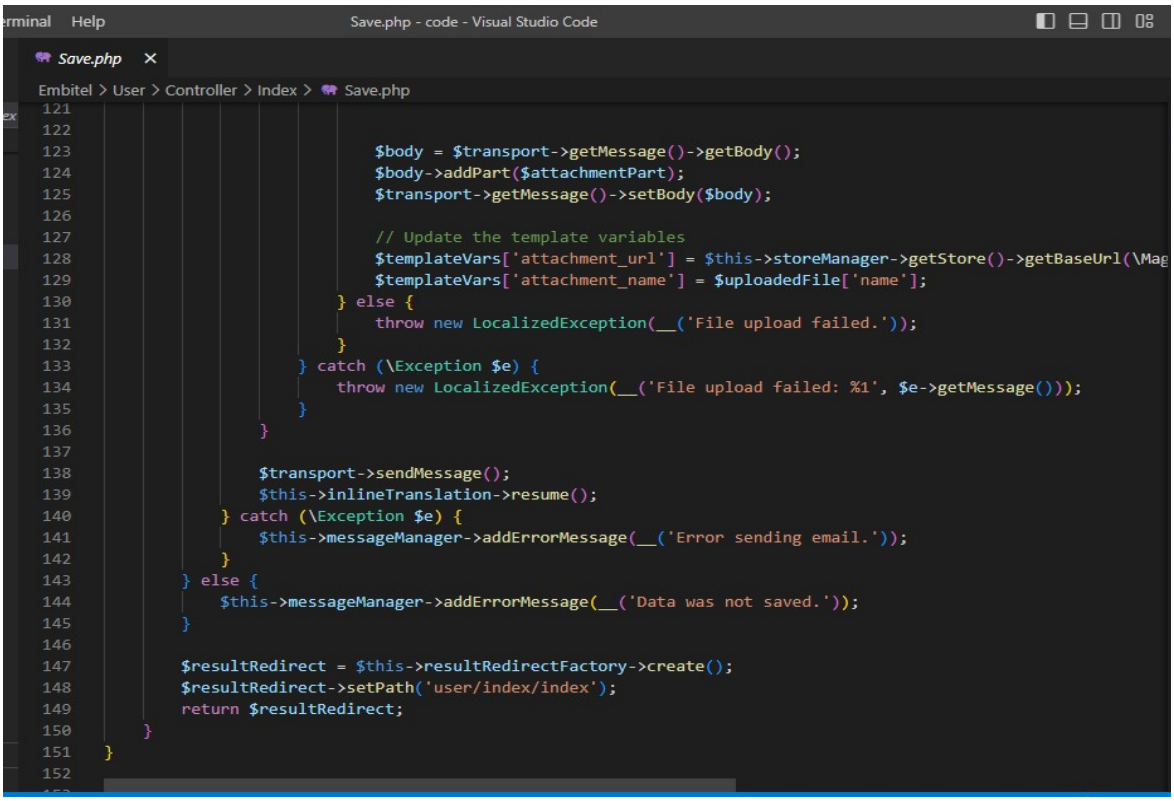



minal Help
Save.php - code - Visual Studio Code
Save.php
Embitel > User > Controller > Index > Save.php
StoreManagerInterface $storeManager,
\Magento\MediaStorage\Model\File\UploaderFactory $fileUploaderFactory,
Filesystem $filesystem
) {
$this->testFactory = $testFactory;
$this->transportBuilder = $transportBuilder;
$this->inlineTranslation = $inlineTranslation;
$this->storeManager = $storeManager;
$this->fileUploaderFactory = $fileUploaderFactory;
$this->filesystem = $filesystem;
parent::__construct($context);
}
public function execute()
{
$request_data = $this->getRequest()->getParams();
$uploadedFile = $this->getRequest()->getFiles('attachment-pdf');
//print_r($uploadedFile["name"]);die;
$data = [
'name' => $request_data['name'],
'email' => $request_data['email'],
'phone' => $request_data['phone'],
'location' => $request_data['location'],
'attachment-pdf' => $uploadedFile["name"],
'date_of_birth' => $request_data['date_of_birth'],
];
$testFactory = $this->testFactory->create();
$test = $testFactory->setData($data);
$test->save();


minal Help
Save.php - code - Visual Studio Code
Save.php
Embitel > User > Controller > Index > Save.php
$test = $testFactory->setData($data);
$test->save();
if ($test->getUser_id()) {
$this->messageManager->addSuccessMessage(__('Your Application has been sent.'));
// Send email to the customer
$templateVars = [
'data' => $data,
'subject' => 'Job Application', // Set the subject
'attachment_url' => '',
'attachment_name' => '',
];
// $from = ['email' => 'shashankganesh53@gmail.com', 'name' => 'Shashank'];
// $to = [$data['email'], $data['name']];
$from = ['email' => $data['email'], 'name' => $data['name']];
$to = ['shashankganesh53@gmail.com', 'Shashank'];
$storeId = $this->storeManager->getStore()->getId();
try {
$storeScope = \Magento\Store\Model\ScopeInterface::SCOPE_STORE;
$templateOptions = [
'area' => \Magento\Framework\App\Area::AREA_FRONTEND,
'store' => $storeId,
];
$this->inlineTranslation->suspend();
$transport = $this->transportBuilder
->setTemplateIdentifier('user_confirmation')

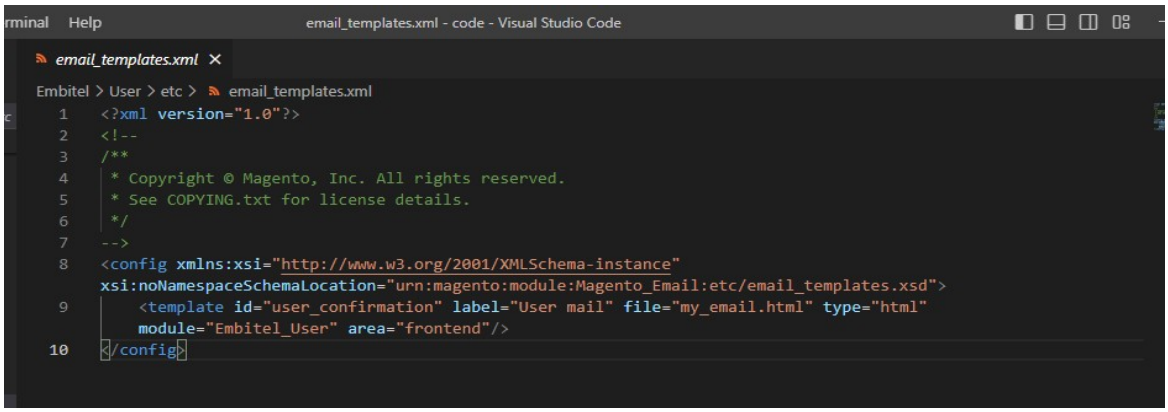



```
92              $this->inlineTranslation->suspend();
93              $transport = $this->transportBuilder
94                  ->setTemplateIdentifier('user_confirmation')
95                  ->setTemplateOptions($templateOptions)
96                  ->setTemplateVars($templateVars)
97                  ->setFrom($from)
98                  ->addTo($to)
99                  ->getTransport();
100
101             // Get the uploaded file information
102             if ($uploadedFile && !empty($uploadedFile['name'])) {
103                 try {
104                     $uploader = $this->fileUploaderFactory->create(['fileId' => 'attachment-pdf']);
105                     $uploader->setAllowedExtensions(['pdf']);
106                     //$uploader->setAllowRename(true);
107                     $uploader->setFilesDispersion(true);
108                     $destinationPath = $this->filesystem->getDirectoryRead(DirectoryList::MEDIA)->getAbsol
109                     $result = $uploader->save($destinationPath);
110
111                     if ($result['file']) {
112                         $attachmentFile = $result['file'];
113                         $attachmentFilePath = $destinationPath . $attachmentFile;
114
115                         // Attach the file to the email
116                         $attachmentPart = new Part(file_get_contents($attachmentFilePath));
117                         $attachmentPart->setType(Mime::TYPE_OCTETSTREAM);
118                         $attachmentPart->setDisposition(Mime::DISPOSITION_ATTACHMENT);
119                         $attachmentPart->setEncoding(Mime::ENCODING_BASE64);
120                         $attachmentPart->setFileName($uploadedFile['name']);
121
122
123                         $body = $transport->getMessage()->getBody();
```

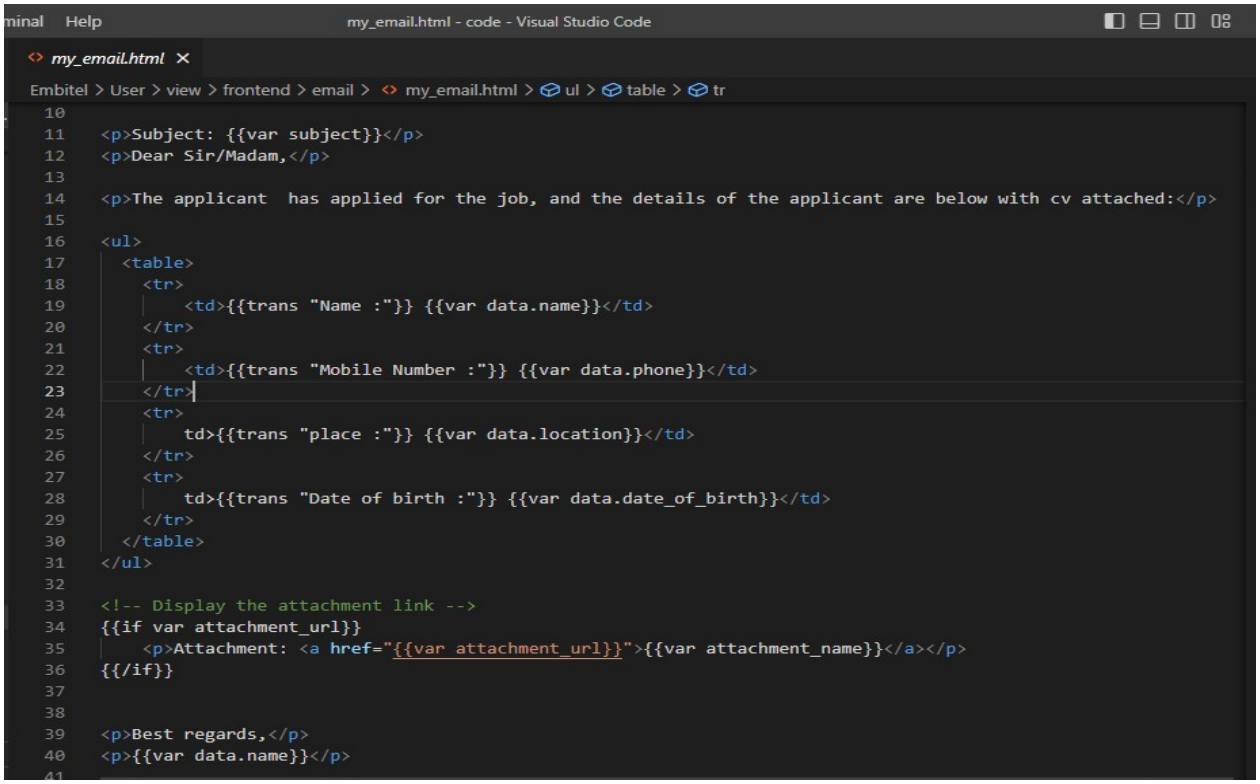



```
121
122
123                         $body = $transport->getMessage()->getBody();
124                         $body->addPart($attachmentPart);
125                         $transport->getMessage()->setBody($body);
126
127                         // Update the template variables
128                         $templateVars['attachment_url'] = $this->storeManager->getStore()->getBaseUrl(\Mag
129                         $templateVars['attachment_name'] = $uploadedFile['name'];
130                     } else {
131                         throw new LocalizedException(__('File upload failed.'));
132                     }
133                 } catch (\Exception $e) {
134                     throw new LocalizedException(__('File upload failed: %1', $e->getMessage()));
135                 }
136             }
137
138             $transport->sendMessage();
139             $this->inlineTranslation->resume();
140         } catch (\Exception $e) {
141             $this->messageManager->addErrorMessage(__('Error sending email.'));
142         }
143     } else {
144         $this->messageManager->addErrorMessage(__('Data was not saved.'));
145     }
146
147     $resultRedirect = $this->resultRedirectFactory->create();
148     $resultRedirect->setPath('user/index/index');
149     return $resultRedirect;
150 }
151 }
152
```

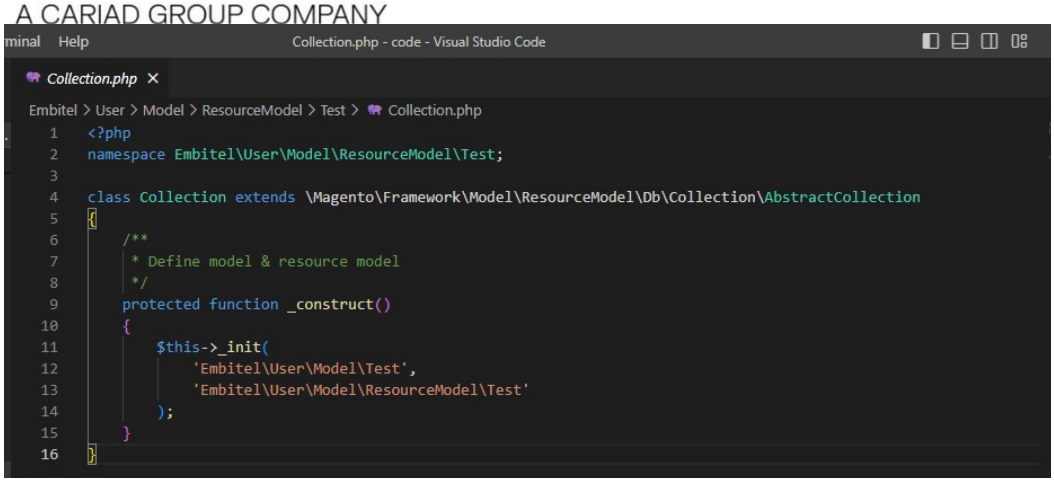
1. For sending mail we need email_templates.xml which is placed inside etc



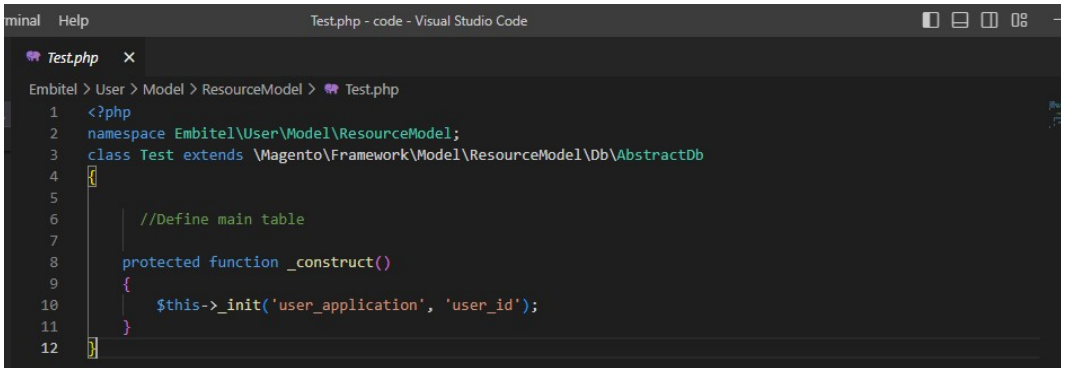
- And to send a attachment and email content to a reciver we need to create a html file view/frontend/email/my_email.html



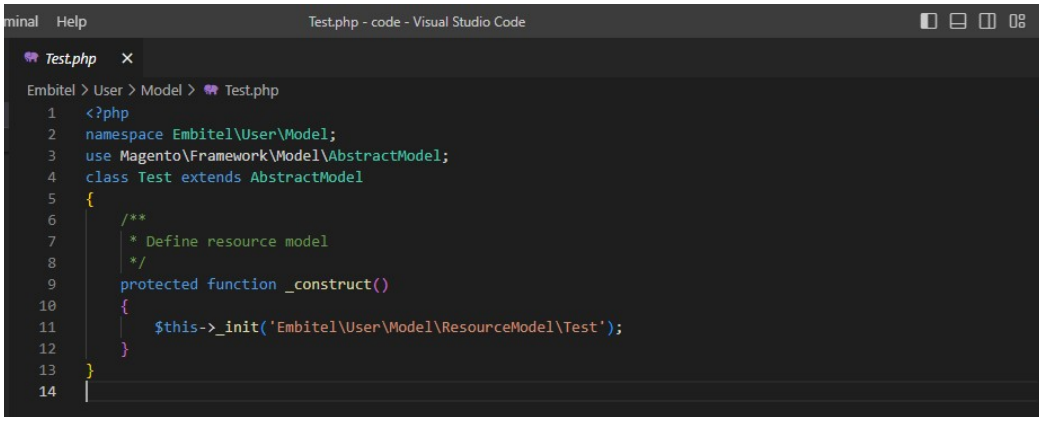
- And for saving the data in db table we need to create a model file and inside Resorcemodel/Test/Collection.php

- 

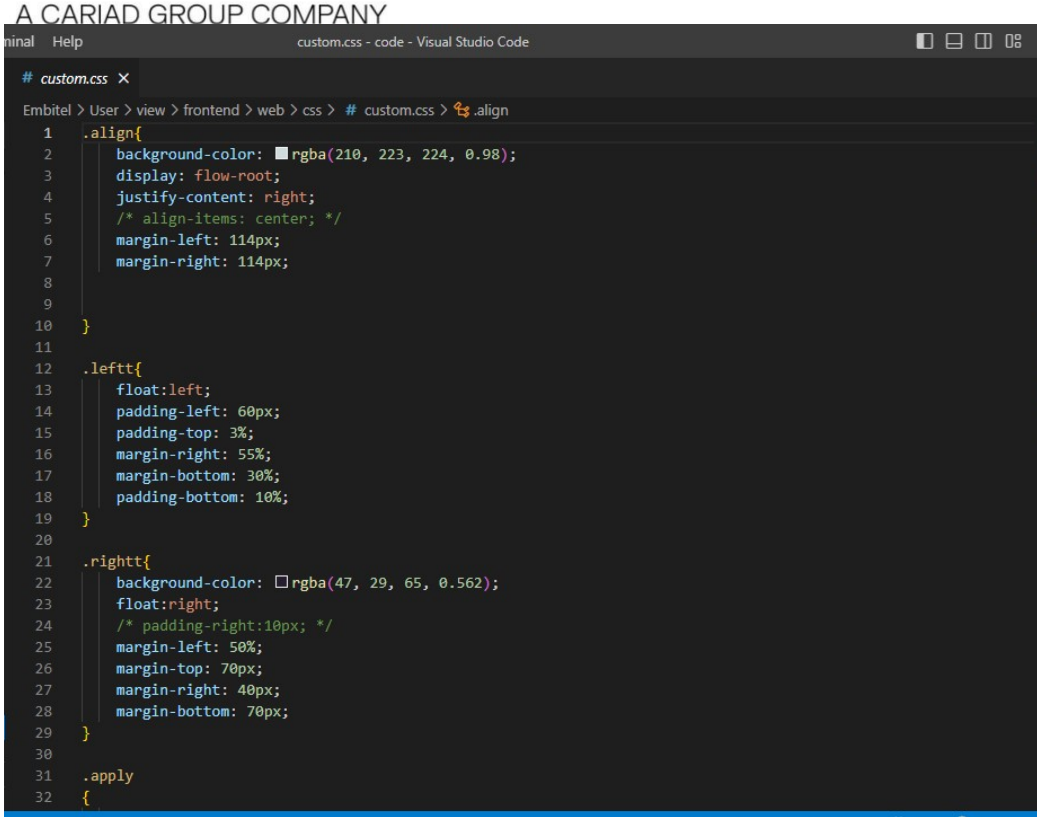
- Inside Resourcemodel test.php

- 

- Inside Model test.php

- 

- For adding css I created custom.css file the path for that is
- View\frontend\web\css\custom.css
-


inal Help
custom.css - code - Visual Studio Code
custom.css
Embitel > User > view > frontend > web > css > # custom.css > .align
.align{
background-color: rgba(210, 223, 224, 0.98);
display: flow-root;
justify-content: right;
/* align-items: center; */
margin-left: 114px;
margin-right: 114px;
}
.leftt{
float:left;
padding-left: 60px;
padding-top: 3%;
margin-right: 55%;
margin-bottom: 30%;
padding-bottom: 10%;
}
.rightt{
background-color: rgba(47, 29, 65, 0.562);
float:right;
/* padding-right:10px; */
margin-left: 50%;
margin-top: 70px;
margin-right: 40px;
margin-bottom: 70px;
}
.apply
{

➢


embitel
A CARIAD GROUP COMPANY


embitel
A CARIAD GROUP COMPANY