**Developer Guide**

Prepared for

**AgentPortal**

VERSION 1.0

DATE: 18-Oct-2021

Prepared by

HazelSoft

Contents

[1. Introduction 2](#_Toc85475934)

[2. Backend 2](#_Toc85475935)

[2.1 Seeding screen and its elements 2](#_Toc85475936)

[2.2 Create Model in database 3](#_Toc85475937)

[2.2.1 Before Scaffolding Models 4](#_Toc85475938)

[2.2.2 Scaffolding Models 4](#_Toc85475939)

[2.2.3 After Scaffolding Models 4](#_Toc85475940)

[2.3 Create Company Controller 5](#_Toc85475941)

[3. Front End 8](#_Toc85475942)

[3.1 AgentPortal 9](#_Toc85475943)

[3.1.1 ActionBar 9](#_Toc85475944)

[3.1.2 FilterComponent 10](#_Toc85475945)

[3.1.3 QueryMsg 13](#_Toc85475946)

[3.1.4 Table 13](#_Toc85475947)

[3.1.5 Pagination 15](#_Toc85475948)

[3.1.6 Result from 16](#_Toc85475949)

[3.1.7 Delete Modal 16](#_Toc85475950)

[3.1.8 Scripts 17](#_Toc85475951)

[3.1.9 Company Adds New Modal. 17](#_Toc85475952)

[3.1.10 Company Modal body 18](#_Toc85475953)

[3.1.11 Company Modal Form 18](#_Toc85475954)

[3.1.12 Submit Form 19](#_Toc85475955)

[3.2 Intelli.PortalServices 19](#_Toc85475956)

[4. Creating a New Screen 20](#_Toc85475957)

[4.1 Adding View to the project 20](#_Toc85475958)

[4.2 Adding Controller to the project 20](#_Toc85475959)

[4.3 Adding the Table in the Screen. 20](#_Toc85475960)

[4.4 Adding action bar in new Screen. 21](#_Toc85475961)

[4.5 Adding pagination in Table 22](#_Toc85475962)

[4.6 Adding JavaScript Files in Project. 23](#_Toc85475963)

[ This document ends here 23](#_Toc85475964)

# 

# Introduction

This document guides developer how to implement a screen with complete steps involved in backend and frontend.

This document assumes that development enviornment for both backend and frontend is already set and all the required framework and libraries are installed as per instructions in backend and frontend documentation.

As an example we’ll create company screen.

# Backend

## Seeding screen and its elements

Seed screen and its elements by creating and running migration as follows:

USE [AgentPortal]

GO

IF NOT EXISTS (SELECT \*

FROM [dbo].[\_\_EFMigrationsHistory]

WHERE [MigrationId] = '20211008221300\_Seed\_Screen\_Data')

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

SET IDENTITY\_INSERT [dbo].[Screens] ON

INSERT INTO [dbo].[Screens] ([Id], [ScreenName], [IsActive], [CreatedAt], [UpdatedAt])

VALUES (12, 'Company', 1, 0, 0)

SET IDENTITY\_INSERT [dbo].[Screens] OFF

insert into ScreenElements values ('CompanyScreenElement1', 12, 1, 0, 0)

insert into ScreenElements values ('CompanyScreenElement2', 12, 1, 0, 0)

insert into ScreenElements values ('CompanyScreenElement3', 12, 1, 0, 0)

INSERT INTO [\_\_EFMigrationsHistory] ([MigrationId], [ProductVersion])

VALUES (N'20211008221300\_Seed\_Screen\_Data', N'5.0.9');

COMMIT;

PRINT 'Data Seeded Successfully!'

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION

PRINT 'Error: ' + ERROR\_MESSAGE()

PRINT 'Transactions Rolled Back'

END CATCH

END

## Create Model in database

Create database table and perform following steps to scaffold model.

In order to use repository pattern for new models, please perform following steps:

### Before Scaffolding Models

* Database table should have Id of type int as its primary key
* IsActive column of type boolean
* CreatedAt column of type bigint
* UpdatedAt column of type bigint

### Scaffolding Models

* To scaffold models when changes are made in database, use following command:

scaffold-dbcontext "Data Source=.\SQLEXPRESS;Initial Catalog=AgentPortal;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Model -ContextDir Database -Project Intelli.AgentPortal.Domain -NoOnConfiguring -Force -t AspNetUsers, Batches, ColumnPreferences, configurations, Countries, DocumentClasses, DocumentGroupNames, DocumentsPerCompanies, PasswordHistory, RoleScreens, RoleScreenColumns, RoleScreenElements, Screens, ScreenColumns, ScreenElements, SystemRoles, SystemUsers, SystemUserCountries, SystemUserRoles, UserPreferences, UserSessions, UsersPerCompanies,

Companies

* If new tables are added in database it should be appended in above mentioned command at the end e.g. ,table\_name1, table\_name2

After running above mentioned command you have to do following steps to remove errors:

* Remove AspNetUsers model class as it is already added in custom folder to be inherited from IdentityUser base class.
* Generated AgentPortalContext class should not be inherited from DbContext because it is being inherited by IdentityDbContext in its partial implementation which is required to use identity framework. So remove DbContext inheritence from generated AgentPortalContext class.

### After Scaffolding Models

Now inside Intelli.AgentPortal.Domain => Model => Custom folder of domain library, add partial class with the same name as of model name. This partial class should implement IEntity interface.

AgentPortal\Intelli.AgentPortal.Domain\Model\Core\IEntity.cs

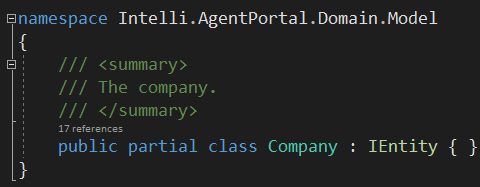


Figure 1

## Create Company Controller

Create company controller as follows:

using AutoMapper;

using Intelli.AgentPortal.Api.DTO;

using Intelli.AgentPortal.Domain.Core.Helpers;

using Intelli.AgentPortal.Domain.Core.Repository;

using Intelli.AgentPortal.Domain.Database;

using Intelli.AgentPortal.Domain.Model;

using Intelli.AgentPortal.EventBus.RabbitMQ.Event;

using Intelli.AgentPortal.EventBus.RabbitMQ.Sender;

using Intelli.AgentPortal.Shared;

using Intelli.AgentPortal.Shared.Mvc.Controllers;

using Intelli.AgentPortal.Shared.Mvc.Resources;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace Intelli.AgentPortal.Api.Controllers.v1

{

/// <summary>

/// The Companies Controller.

/// </summary>

[Route("api/v1/[controller]")]

[ApiController]

public class CompaniesController : BaseController

{

private readonly IMapper \_mapper;

private readonly ILogger \_logger;

private readonly IRepository<Company> \_repository;

/// <summary>

/// Initializes a new instance of the <see cref="CompaniesController"/> class.

/// </summary>

/// <param name="context">Instance of <see cref="AgentPortalContext"/> will be injected</param>

/// <param name="mapper">Instance of <see cref="IMapper"/> will be injected</param>

/// <param name="logger">Instance of <see cref="ILogger"/> will be injected</param>

/// <param name="sender">Instance of <see cref="IEventSender"/> will be injected</param>

public CompaniesController(AgentPortalContext context,

IMapper mapper,

ILogger<CompaniesController> logger,

IEventSender sender)

{

\_repository = new GenericRepository<Company>(context);

((GenericRepository<Company>)\_repository).AfterSave = (logs) =>

sender.SendEvent(new MQEvent<List<AuditEntry>>(AUDIT\_EVENT\_NAME, (List<AuditEntry>)logs));

\_logger = logger;

\_mapper = mapper;

}

/// <summary>

/// Get All Companies.

/// </summary>

/// <returns>List of CompanyDTO</returns>

[HttpGet("all")]

public async Task<IActionResult> GetAll()

{

var dto = new CompanyDTO();

var result = await \_repository.GetAllActiveAsync(nameof(dto.CompanyName));

return Ok(new { Items = result.List.Select(x => \_mapper.Map<CompanyDTO>(x)).ToList() });

}

/// <summary>

/// Gets the companies w.r.t quey string parameters.

/// </summary>

/// <param name="queryStringParams">The query string params.</param>

/// <returns>An IActionResult.</returns>

[HttpGet]

public IActionResult Get([FromQuery] QueryStringParams queryStringParams)

{

\_logger.LogInformation("Get Companies Called with params: {0}", queryStringParams);

PagedResult<CompanyDTO> result = null;

try

{

QueryResult<Company> queryResult = \_repository.Get(

queryStringParams.FilterExpression,

queryStringParams.OrderBy,

queryStringParams.PageSize,

queryStringParams.PageNumber);

int total = queryResult.Count;

IEnumerable<Company> roleList = queryResult.List;

result = new PagedResult<CompanyDTO>(

total,

queryStringParams.PageNumber,

roleList.Select(x => \_mapper.Map<CompanyDTO>(x)).ToList(),

queryStringParams.PageSize

);

}

catch (ArgumentException e)

{

// Log error message

\_logger.LogError("{0}: {1}", e.Message, e);

return BadRequest(new

{

Errors = e,

e.Message

});

}

return Ok(result);

}

/// <summary>

/// Creates the company.

/// </summary>

/// <param name="dto">The company dto.</param>

/// <returns>An IActionResult.</returns>

[HttpPost]

public IActionResult Post(CompanyDTO dto)

{

// Checking if the passed DTO is valid

if (!ModelState.IsValid || dto == null)

return BadRequest(MsgKeys.InvalidInputParameters);

var entity = \_mapper.Map<Company>(dto);

\_repository.Insert(entity);

\_repository.SaveChanges(UserName, null);

return Ok(\_mapper.Map<CompanyDTO>(entity));

}

/// <summary>

/// Gets the company.

/// </summary>

/// <param name="id">The company id.</param>

/// <returns>The CompanyDTO.</returns>

[HttpGet("{id}")]

public async Task<IActionResult> Get(int id)

{

var query = \_repository.Query(x => x.Id == id)

.Include(x => x.DocumentsPerCompanies)

.Include(x => x.SystemUsers);

var entity = await query.FirstOrDefaultAsync();

if (entity == null) return NotFound();

var dto = \_mapper.Map<CompanyDTO>(entity);

return Ok(dto);

}

/// <summary>

/// Updates the company.

/// </summary>

/// <param name="id">The company id.</param>

/// <param name="dto">The CompanyDTO.</param>

/// <returns>An IActionResult.</returns>

[HttpPut("{id}")]

public IActionResult Put(int id, CompanyDTO dto)

{

// Checking if the passed DTO is valid

if (!ModelState.IsValid || dto == null || id != dto.Id)

return BadRequest(MsgKeys.InvalidInputParameters);

var entity = \_mapper.Map<Company>(dto);

\_repository.Update(entity);

\_repository.SaveChanges(UserName, null);

return Ok(\_mapper.Map<CompanyDTO>(entity));

}

/// <summary>

/// Delete the company.

/// </summary>

/// <param name="id">The company id.</param>

/// <returns>An IActionResult.</returns>

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(int id)

{

var query = \_repository.Query(x => x.Id == id)

.Include(x => x.DocumentsPerCompanies)

.Include(x => x.SystemUsers);

var entity = await query.FirstOrDefaultAsync();

if (entity == null) return NotFound();

if (entity.DocumentsPerCompanies.Count == 0 && entity.SystemUsers.Count == 0)

{

entity.IsActive = false;

\_repository.SaveChanges(UserName, null);

return Ok(MsgKeys.DeletedSuccessfully);

}

return BadRequest(MsgKeys.ChildEntityExists);

}

}

}

# Front End

The entire Solution of agent portal UI consists of the following two projects.

* **AgentPortal:** This project contains all UI-related work.
* **Intelli.PortalServices:** This project contains all API-related work.

Below there is an explanation of both project structure.

## AgentPortal

We will take the screen of companies to understand the code structure of the agent portal project. Below are the details of the company View structure.

### ActionBar

For reference, if we first take the Companies>Index.cshtml file the first thing we see on the page is the action bar partial view as given below.

|  |
| --- |
| <**partial** **name**="~/Views/Shared/\_ActionBar.cshtml" **view-data**="ViewData" /> |

After importing the action bar partial component, you will have the following functionalities:

* **Export Data:** for exporting data in pdf or excel form
* **Add New button:** for adding a new record to the server.
* **Filter:** for filtering the data coming from the server.

The important thing for the action bar is the viewData which gives details to the action bar for which purpose we are using it or for what screen we are using it. Following are the details which you have to send the action bar to make it functional:

* **ScreenConfigurations**: Screen configuration is an object that contains the configurations of the screen. We can set it from the controller method (AgentPortal>Controllers> CompaniesController.cs) and pass it to the parent view as view data and which will be passed to the action bar as a partial view.

|  |
| --- |
| // set the screen configuration  ScreenConfiguration screenConfiguration = new ScreenConfiguration(ScreenNamesConstant.COMPANY, true, true, true, true, true ,true, true); |

The above object is initializing the following constructor for the screen configurations which is available at the following path.

Path: AgentPortal>Helpers>Configurations> ScreenConfiguration.cs

|  |
| --- |
| public ScreenConfiguration(string ScreenName,  bool IsExportEnable = false,  bool IsFilterEnable = false,  bool IsAddItemsEnable = false,  bool IsSortEnable = false,  bool IsPaginationEnable = false,  bool IsEditable= false,  bool IsDeleteAble = false)  {  this.ScreenName = ScreenName;  this.IsExportEnable = IsExportEnable;  this.IsFilterEnable = IsFilterEnable;  this.IsAddItemsEnable = IsAddItemsEnable;  this.IsSortEnable = IsSortEnable;  this.IsPaginationEnable = IsPaginationEnable;  this.IsEditable = IsEditable;  this.IsDeleteAble = IsDeleteAble;  } |

The Action bar will show elements according to given configurations of the screen and authorization will apply upon the rendering of the action bar component for example if the filter is enabled in screen configuration and according to the user role the user is not allowed to use the filter component then it will not render and same in case if a user is allowed to use the filter from server role but you have set false for the filter in screen configuration then filter will not render.

* **FileName:** file name contains the name of the file you want to set for exporting the file.

|  |
| --- |
| ViewData[MsgKeys.FileName] = "Company"; |

The above view data is given as a reference from the file which path is given below.

Path: AgentPortal>Views>Companies>Index.cshtml

* **Controller:** you have to give the controller name here so that the action bar will perform actions from the given controller.

|  |
| --- |
| ViewData[MsgKeys.Controller] = "Companies"; |

The above view data is given as a reference from the file which path is given below.

Path: AgentPortal>Views>Companies>Index.cshtml

* **ExportUrl:** you have to provide the export method name from the controller which is responsible for getting all data related to the company from the controller.

|  |
| --- |
| ViewData[MsgKeys.ExportUrl] = "getCompaniesAllDataForExport"; |

The above view data is given as a reference from the file which path is given below.

Path: AgentPortal>Views>Companies>Index.cshtml

* **FilterModel**: you have to provide the filter Model in the view data which contains the filter DTO which can be used by the filter component for filtering the data

|  |
| --- |
| FilterDTO filterDTO = new FilterDTO();  ViewData[MsgKeys.FilterModel] = filterDTO; |

The above object will pass from the controller (AgentPortal>Controllers> CompaniesController.cs) method to view which again passes from parent view to partial view.

For more detail kindly check out the companies controller and companies' view the path of both files are given above in explanations.

### FilterComponent

The next thing you will see after the action bar is the partial view of the filter component which is being rendered based on the screen configuration that is set by the controller method and user role authorization. The snippet is given below.

|  |
| --- |
| <!--if the filter is enable in audit screen screen configration and user has the access to use the filter then render the filter component partial view -->  @if (screenConfigurations.IsFilterEnable && AuthorizationUtil.IsUserHasScreenElementPrivilege(screenConfigurations.ScreenName, ScreenElementsConstants.FILTER, Context)  && !AuthorizationUtil.IsUserHasNoAccessScreenElementPrivilege(screenConfigurations.ScreenName, ScreenElementsConstants.FILTER, Context))  {  <!--filter component partial view and view data is pass as a parameters -->  <**partial** **name**="~/Views/Shared/\_filterComponent.cshtml" **view-data**="ViewData" />  } |

The above partial view is also using the viewData that is coming from the controller method and the viewData that is set by the parent view itself. Below are the details of the view data that is required by the filter component to make the filter fully functional.

* **FilterData:**  This filtered data is the columns name list that is used by the filter component to display in the select tag. The snippet and screenshot are given below.

|  |
| --- |
| var data = ViewData[MsgKeys.FilterData] as List<FilterColumnsDTO>; |

|  |
| --- |
|  |

**Fig 3: usage of filterData in ui**

This filtered data is passed by the controller method to parent view which is again passed to partial view. As above snippet the filter columns partial view is using this as List<FilterColumnsDTO>; This Dto is coming from the portal services project which is discussed in the postal service part. The path for FilterColumnsDto is given below:

Path: Intell.PortalServices>DTO >Companies> FilterColumnsDto.cs

This data is passing to the filter js script to show and run the filter functionalities the snippet for passing the filter data to scripts is given below:

|  |
| --- |
| @if (data != null)  {  <script>  //pass the table columns data to js script if there is no data pass the empty list just  let columns = JSON.parse('@Json.Serialize(@data)')  let url = '@url'  </script>  }  else  {  <script>  //pass the table columns data to js script if there is no data pass the empty list just  let columns = []  </script>  }  <script src="~/js/Shared/Filter.js"></script> |

As it is clear from the above snippet that data is setting in columns variable which is being used by filter.js script file. The path where filter.js is available is given below.

Path: AgentPortal>wwwroot>js>Shared>Filter.js

* **FilterModel:** you have to provide the filter Model in the view data which contains the filter DTO which can be used by the filter component

|  |
| --- |
| FilterDTO filterDTO = new FilterDTO();  ViewData[MsgKeys.FilterModel] = filterDTO; |

The above object will pass from the controller method to view which again passes from parent view to partial view. the path for the filter DTO is given in the above explanation.

* **Controller:** you have to give the controller name here so that the action bar will perform actions from the given controller.

|  |
| --- |
| ViewData[MsgKeys.Controller] = "Companies"; |

The above view data is given as a reference from the file which path is given below.

Path: AgentPortal>Views>Companies>Index.cshtml

* **ExportUrl:** you have to provide the export method name from the controller which is responsible for getting all data related to the company from the controller.

|  |
| --- |
| ViewData[MsgKeys.ExportUrl] = "getCompaniesAllDataForExport"; |

The above view data is given as a reference from the file which path is given below.

Path: AgentPortal>Views>Companies>Index.cshtml

* **ScreenConfigurations:** screen configuration is the same as we explained earlier in the action bar section.
* **RelationalData:** this view data is optional if there is a column that is relational like in the user's screen the company column is relational we have to pass the company list there so it is optional.

### QueryMsg

The next thing you will see after the filter component is the partial view of the Query msg which is being rendered based upon the filterDTO which is coming from the controller if filterDto contains the query msg in it it will render the filter msg there the snippet and screenshot are given below.

|  |
| --- |
| <!-- if there is a message from the filer and render message on the screen -->  @if (queryMsg != null && queryMsg.ToString() != "")  {  <**partial** **name**="~/Views/Shared/\_QueryMsg.cshtml" **view-data**="ViewData" />  } |

|  |
| --- |
|  |

**Fig 4: queryMsg in UI**

The partial view needs the following data for functioning properly.

### Table

The next thing you will see after the query msg partial view is the partial view, the table has been used along with pagination. Every screen has its table and table components will generate based on the Columns util. for the Company screen, we have to import the columns util in the HTML file as given below.

|  |
| --- |
| @using AgentPortal.Helpers.Utils |

The above URL will import the columns util this contains the list of the table columns we are using to display the data to the screen. Below is the list of the columns we are using to render the user table columns

|  |
| --- |
| public static List<FilterColumnsDTO> CompaniesTableColumns = new List<FilterColumnsDTO>  {  new FilterColumnsDTO(ColumnKeys.CompanyName , FilterTypesConstants.TEXT),  new FilterColumnsDTO(ColumnKeys.CallBackUrl , FilterTypesConstants.TEXT),  new FilterColumnsDTO(ColumnKeys.Slaimportance , FilterTypesConstants.NUMERIC),  new FilterColumnsDTO(ColumnKeys.Email , FilterTypesConstants.TEXT),  new FilterColumnsDTO(ColumnKeys.IsSignedCompany , FilterTypesConstants.BOOLEAN),  new FilterColumnsDTO(ColumnKeys.SendRejectionReasonAsCode , FilterTypesConstants.BOOLEAN),  new FilterColumnsDTO(ColumnKeys.SendLink , FilterTypesConstants.BOOLEAN),  new FilterColumnsDTO(ColumnKeys.SupportsCalls , FilterTypesConstants.BOOLEAN),  new FilterColumnsDTO(ColumnKeys.VideoCallBackUrl , FilterTypesConstants.TEXT),  new FilterColumnsDTO(ColumnKeys.IsActive , FilterTypesConstants.BOOLEAN)  }; |

The above snippet is available on the following path.

Path: AgentPortal>Helpers>Utils> ColumnUtil.cs

Every object in the usertablescolumns list is initializing the following constructor given below:

|  |
| --- |
| public FilterColumnsDTO(string Name , string Type , string RelationType = null)  {  this.Name = Name;  this.Type = Type;  this.RelationType = RelationType;  } |

In the above constructor, we are using the following details given below.

* + **Name** the name of the column.
  + **Type:** defines the type of columns that can be used in the filter to show the type of field.
  + **RelationType**: it contains the relationship name you for the relational types fields default value will be null.

The above Dto is available on the following path.

Path: Intelli.portalServices>Dto>FilterColumnsDTO.cs

The controller sends data to view which is being render on the screen as the table row the snippet to render the row data is given below.

For applying the pagination you have to import the pagination partial view in the in the parent view component and pass the pagination view data in it. Below is the partial view given which is imported in the role index.cshtml file.

|  |
| --- |
| <!--if the paggination is enable in role screen screen configration and user has the access to use the pagination then render the pagination component partial view->  @if (screenConfigurations.IsPaginationEnable && AuthorizationUtil.IsUserHasScreenElementPrivilege(screenConfigurations.ScreenName, ScreenElementsConstants.PAGGINATION, Context)  &&!AuthorizationUtil.IsUserHasNoAccessScreenElementPrivilege(screenConfigurations.ScreenName, ScreenElementsConstants.PAGGINATION, Context))  {  <**partial** **name**="~/Views/Shared/\_Pagination.cshtml" **view-data**="ViewData" />  } |

Following view, data is required by the pagination partial component.

* + **PaginationData:** pagination data contains all the details about the page its size, current page, etc. it will be used to show the pagination on screen.

|  |
| --- |
| ViewData[MsgKeys.PaginationData] as IPagedResultDTO |

* + **QueryMsg:** it contains the query msg if any which can be pass again to the controller.

|  |
| --- |
| ViewData[MsgKeys.QueryMsg]; |

* + **FilterModel:** you have to provide the filter Model in the view data which contains the filter DTO which can be used by the filter component

|  |
| --- |
| FilterDTO filterDTO = new FilterDTO();  ViewData[MsgKeys.FilterModel] = filterDTO; |

The above object will pass from the controller method to view which again passes from parent view to partial view.

* + **ScreenConfigurations:** a screen configuration is an object that contains the configurations of the screen. We can set it from the controller method and pass it to the view as view data.

|  |
| --- |
| ScreenConfiguration screenConfiguration = new ScreenConfiguration(ScreenNamesConstant.COMPANY, true, true, true, true, true, true, true); |

### Pagination

For applying the pagination you have to import the pagination partial view in the in the parent view component and pass the pagination view data in it. Below is the partial view given which is imported in the role company index.cshtml file

|  |
| --- |
| <!--if the paggination is enable in role screen screen configration and user has the access to use the pagination then render the pagination component partial view->  @if (screenConfigurations.IsPaginationEnable && AuthorizationUtil.IsUserHasScreenElementPrivilege(screenConfigurations.ScreenName, ScreenElementsConstants.PAGGINATION, Context)  &&!AuthorizationUtil.IsUserHasNoAccessScreenElementPrivilege(screenConfigurations.ScreenName, ScreenElementsConstants.PAGGINATION, Context))  {  <**partial** **name**="~/Views/Shared/\_Pagination.cshtml" **view-data**="ViewData" />  } |

Following view, data is required by the pagination partial component.

* + **PaginationData:** pagination data contains all the details about the page its size, current page, etc. it will be used to show the pagination on screen.

|  |
| --- |
| ViewData[MsgKeys.PaginationData] as IPagedResultDTO |

* + **QueryMsg:** it contains the query msg if any which can be pass again to the controller.

|  |
| --- |
| ViewData[MsgKeys.QueryMsg]; |

* + **FilterModel:** you have to provide the filter Model in the view data which contains the filter DTO which can be used by the filter component

|  |
| --- |
| FilterDTO filterDTO = new FilterDTO();  ViewData[MsgKeys.FilterModel] = filterDTO; |

The above object will pass from the controller method to view which again passes from parent view to partial view.

* + **ScreenConfigurations:** a screen configuration is an object that contains the configurations of the screen. We can set it from the controller method and pass it to the view as view data.

|  |
| --- |
| ScreenConfiguration screenConfiguration = new ScreenConfiguration(ScreenNamesConstant.COMPANY, true, true, true, true, true, true, true); |

For more reference check out the above subheadings explanations or check out the controller of the company in code.

### Result from

If the screen contains filter, sort, or pagination-related operations then we have to add the resulting form at the end of the component the sorting, filter, and pagination operations depend upon this form. The following snippet for result form is given bellow:

|  |
| --- |
| <!-- form to be submitted in the case of paggination and sortingh -->  <form hidden id="resultForm" style="display:none;" method="post" **asp-controller**="Companies" **asp-action**="Index">  <input id="FilterString" **type**="text" **asp-for**="FilterString" />  <input id="FilterMessage" **type**="text" **asp-for**="FilterMessage" />  <input id="CurrentPage" **type**="text" **asp-for**="CurrentPage" />  <input id="PreviousPage" **type**="text" **asp-for**="PreviousPage" />  <input id="SortedColumn" **type**="text" **asp-for**="SortedColumn" />  <input id="FilterType" **type**="text" **asp-for**="SortType" />  </form> |

### Delete Modal

If there is a delete operation present on the screen then you have to add delete alert modal partial view on the screen to show an alert on every delete button click. The following snippet for the delete modal is given below:

|  |
| --- |
| <**partial** **name**="~/Views/Shared/\_DeleteAlertModal.cshtml" **view-data**="ViewData" />  <input type="hidden" id="selectedRow" class="d-none" /> |

In the above snippet, selected row input is important as it is saving the deleted row id, and the delete alert modal is using it to delete the selected row.

### Scripts

At the end of the document, we have to pass all kinds of the script which is used by the cshtml file the reference snippet is given below which is taken from the company index.cshtml file.

|  |
| --- |
| <!-- if Error show alert that it is a server Error -->  <script src="~/lib/sweetalert2/sweetalert2.min.js"></script>  <script src="~/js/utils/ToastUtil.js"></script>  @if (TempData[MsgKeys.ServerError] != null || TempData[MsgKeys.ApplicationError] != null || TempData[MsgKeys.Message] != null)  {  <script type="text/javascript">  let success = []  if ('@TempData[MsgKeys.Message]') {  success = '@TempData[MsgKeys.Message]'.split(",")  }  if (success.length > 1) {  toastType.success(success[1])  } else {  toastType.error('@TempData[MsgKeys.ServerError]' ? '@TempData[MsgKeys.ServerError]' :  '@TempData[MsgKeys.ApplicationError]' ? '@TempData[MsgKeys.ApplicationError]' :  '@TempData[MsgKeys.Message]')  }  </script>  }  <script src="~/js/config/Config.js"></script>  <script src="~/js/utils/Constants/ScreenNamesConstant.js"></script>  <script src="~/js/Shared/Utility/ModelCreateEditUtility.js"></script> |

To show error alerts on the screen you have to add the first three scripts to the new screen for more reference check out other views and controllers.

### Company Adds New Modal.

Add new modal pops up when the Add New button is clicked on the action bar. This modal is defined in the action bar and the modal body is pop up from the controller dynamically. Given below is the snippet from the action bar partial view where we have to define the modal partial view and add a button for the company screen

|  |
| --- |
| @if (screenConfiguaration.ScreenName == ScreenNamesConstant.COMPANY)  {  <!-- check for the read only privileges if readonly access show in disable form else show it in orginal form -->  @if (AuthorizationUtil.IsUserHasReadOnlyScreenElementPrivilege(screenConfiguaration.ScreenName, ScreenElementsConstants.ADDNEW, Context))  {  <button type="button" class="tolbar-items btn btn-secondary disabled">  <i class="fa fa-plus fa-xs"></i> <span class="names-on-smaller-screens">@SharedLocalizer.GetLocalizedHtmlString(LocalizationKeys.AddNew)</span>  </button>  }  {  <button type="button" class="tolbar-items btn btn-secondary" onclick="addRow('@screenConfiguaration.ScreenName')">  <i class="fa fa-plus fa-xs"></i> <span class="names-on-smaller-screens">@SharedLocalizer.GetLocalizedHtmlString(LocalizationKeys.AddNew)</span>  </button>  }  }  <!—modal for the company screen changes dynamically from the controller -->  @if (screenConfiguaration.ScreenName == ScreenNamesConstant.COMPANY)  {  <div class="modal fade" id="modal-add" data-backdrop="static" data-keyboard="false">  <div class="modal-dialog modal-lg modal-dialog-scrollable" role="document">  <**partial** **name**="~/Views/Companies/CompanyModal.cshtml" **view-data**="ViewData" />  </div>  </div>  } |

The View data requirements for the Modal is given below:

* **ScreenConfigurations**: a screen configuration is an object that contains the configurations of the screen. We can set it from the controller method and pass it to the view as view data.

|  |
| --- |
| ScreenConfiguration screenConfiguration = new ScreenConfiguration(ScreenNamesConstant.COMPANY, true, true, true, true, true, true, true); |

### Company Modal body

The modal body is defined in the partial view (CompanyModal.cshtml) given it will be changed according to the data provided to it if a user clicks on the add modal it will generate a new modal for adding the Company data and if the user clicks on the edit of specific Company the specific Company data will be set inside it.

### Company Modal Form

The fields in the Company form are generated based upon the fields utils we have to define all kinds of fields in the fields util class which can be imported in the partial view and it will be rendered accordingly for example for rendering the fields in the company form we have to define all companies fields in the fields util given below.

|  |
| --- |
| // companies modal fields  public static List<FieldsDTO> CompaniesFields = new List<FieldsDTO> {  new FieldsDTO(LocalizationKeys.CompanyName , FieldsTypesConstant.TEXT, ScreenElementsConstants.COMPANY\_NAME\_INPUT , ValidationConstants.CompanyNameValidation),  new FieldsDTO(LocalizationKeys.Email , FieldsTypesConstant.TEXT,ScreenElementsConstants.COMPANY\_EMAIL\_INPUT , ValidationConstants.CompanyEmailValidation),  new FieldsDTO(LocalizationKeys.Slaimportance , FieldsTypesConstant.NUMERIC,ScreenElementsConstants.SALIMPORTANCE , ValidationConstants.SalimportantValidation),    }; |

The above snippet is available on the following path.

Path: AgentPortal>Helpers>Utils>FieldsUtil.cs

Every object in the Company fields list is initializing the following constructor.

|  |
| --- |
| public FieldsDTO(string Name, string Type, string ScreenElementsConstant  , string OnBlurFunction = null)  {  this.Name = Name;  this.Type = Type;  this.ScreenElementsConstant = ScreenElementsConstant;  this.OnBlurFunction = OnBlurFunction;  } |

In the above constructor, we are using the following details given below.

* + **Name:** the name of the field.
  + **Type:** defines the type of field text or password etc.
  + **ScreenElementsConstant**: define the element name which can be used in authorization.
  + **OnBlurFunction**: add function call for field validation which can be used as on blur function default it is null.

### Submit Form

Upon clicking the update or submit button the form will be submitted on the given action method of the controller and in return alert will be shown on the screen.

## Intelli.PortalServices

All API and Dto related work is done in this project to explain this structure we are taking the Companies screen as an example. All companies related data is coming from the CompaniesApi.cs file. Bellow is the path of the file is given

Path: Intelli.portalServices>ApiProxies> CompaniesApi.cs

Below is an example of getting the data of all companies.

|  |
| --- |
| public async Task<ServerResponse> GetAllCompanies(string token)  {  return await base.GetAsync(ApiEndPoints.GetAllCompanies(), token);  } |

The companies controller index method will call this method to get all companies data which is then passed to the View and view can manage to show it accordingly.

|  |
| --- |
| var response = await \_api.GetAllCompanies(UserToken, DataGridSize); |

The above snippet is showing how a controller method is calling the API method to get all company's data.

All DTO is available at the following path

Path: Intelli.portalServices>DTO>

# Creating a New Screen

## Adding View to the project

The View folder contains all of the Views UI. To add the new screen’s UI to the view folder.

Navigate to the following path.

Path: AgentPortal>Views>

Now create a new folder with the view Name and add an index.cshtml razor view in it. This view will render in the middle container on the screen. For example, you want to add a new view of showing

userPerCompany screen. You have to create a new folder in the views folder with the name as UserPerCompany and generate index.cshtml in this folder. For reference check out the Companies folder in code.

## Adding Controller to the project

The Controllers folder contains all of the controllers of the defined Views UI. To add the new View screen’s Controller to the controller folder.

Navigate to the following path.

Path: AgentPortal>Controllers>

Now create a new controller file here. For Example, we want to add the controller for the UserPerCompany screen we will navigate to the controller's folder and add a new controller with the name the UserPerCompanyController.cs the ide will automatically generate a method to show the default view on the screen. First, we need to change the inheritance in our controller to inherit the new controller from the base controller so that we will have to access to use share methods now add authorization, etc. accordingly. For reference check out the Companies controller file in code.

## Adding the Table in the Screen.

To add a table in the Screen we first need to add a column list in the column util class that list will be rendered on-screen afterward. For example, the UserPerCompany screen table requires the user name column or other, etc. we first have to go to the column util class which is available in the following path.

Path: AgentPortal>Helpers>Utils>ColumnsUtil.cs

Then modify this class by making a list name as a UserPerCompanyColumns and add fields accordingly. Below is given the reference list of userPerCompanyColumns. For more reference check out the company table columns. For columnsDTO details read the agent portal section in code structure.

|  |
| --- |
| public static List<FilterColumnsDTO> UserPerCompanyColumns = new List<FilterColumnsDTO>  {  new FilterColumnsDTO(ColumnKeys.CompanyName , FilterTypesConstants.TEXT),  new FilterColumnsDTO(ColumnKeys.IsActive , FilterTypesConstants.BOOLEAN),  }; |

After adding the fields in the FilterColumnsDTO list now we have to render the columns on the screen before rendering you also have to add the localize keys in resource files so that if the user changes his language the language of columns will change accordingly the path for the resource files given below:

Path: AgentPortal>Resources>SharedResources.cs

Upon expanding the shared resources.cs you will get the language-specific resource files to add keys in both files. Also, add the keys as a constant in LocalizationKeys.cs for future use.

Path for the LocalizationKeys file is given below

Path: AgentPortal>Helpers>LocalizationKeys.cs

Now render the added columns in the screen according to columns given to screening for reference check out the companies table headers.

After rendering the columns add authorization in the controller class method and get API data from the server and pass it to view to show it in the server For reference check out Intelli.portalServices section in code structure and also check out companies index.cshtml file in views.

## Adding action bar in new Screen.

The action bar contains the following actions.

* + Export
  + Add new Item
  + Filter

You can hide or show different actions based upon the screen configurations which has been discussed in the code structure section.

For adding the action bar in view just add the following partial view.

|  |
| --- |
| <**partial** **name**="~/Views/Shared/\_ActionBar.cshtml" **view-data**="ViewData" /> |

Set the action bar view Data accordingly which was discussed in the code structure section.

If you are adding add new item model to the current new page then you have to modify the action bar partial view by adding the button click method in the action bar also add a modal body at the end

For example, you are adding the addnewUserpercompany method first you have to add click listener in the action bar as given below.

|  |
| --- |
| @if (screenConfiguaration.ScreenName == ScreenNamesConstant.UserPerCompany)  {  <!-- check for the read only privileges if readonly access show in disable form else show it in orginal form -->  @if (AuthorizationUtil.IsUserHasReadOnlyScreenElementPrivilege(screenConfiguaration.ScreenName, ScreenElementsConstants.ADDNEW, Context))  {  <button type="button" class="tolbar-items btn btn-secondary disabled">  <i class="fa fa-plus fa-xs"></i> <span class="names-on-smallerscreens">@SharedLocalizer.GetLocalizedHtmlString(LocalizationKeys.AddNew)</span>  </button>  }  else  {  <button type="button" class="tolbar-items btn btn-secondary" onclick="addRow('@screenConfiguaration.ScreenName')">  <i class="fa fa-plus fa-xs"></i> <span class="names-on-smaller-screens">@SharedLocalizer.GetLocalizedHtmlString(LocalizationKeys.AddNew)</span>  </button>  }  } |

Now you also have to add the model section, in the end, to show the modal on screen.

|  |
| --- |
| @if (screenConfiguaration.ScreenName == ScreenNamesConstant.UserPerCompany)  {  <div class="modal fade" id="modal-add" data-backdrop="static" data-keyboard="false">  <div class="modal-dialog modal-lg modal-dialog-scrollable" role="document">  <**partial** **name**="~/Views/UserPerCompany/UserPerCompanyModal.cshtml" **view-data**="ViewData" />  </div>  </div>  } |

For more reference check out the Company modal and Company actions in the action bar.

## Adding pagination in Table

You have to set pagination true in screen configuration and also have to add the pagination partial view in the view file and sets pagination View data accordingly. Given below are the configurations for the pagination.

|  |
| --- |
| @if (screenConfigurations.IsPaginationEnable &&  AuthorizationUtil.IsUserHasScreenElementPrivilege(screenConfigurations.ScreenName, ScreenElementsConstants.PAGGINATION, Context)  && !AuthorizationUtil.IsUserHasNoAccessScreenElementPrivilege(screenConfigurations.ScreenName, ScreenElementsConstants.PAGGINATION, Context))  {  <**partial** **name**="~/Views/Shared/\_Pagination.cshtml" **view-data**="ViewData" />  } |

Also add following form at the end of the index.cshtml file

|  |
| --- |
| <form hidden id="resultForm" asp-for ="Index" **asp-controller**="Companies" style="display:none;" method="post">  <input id="FilterString" **type**="text" **asp-for**="FilterString" />  <input id="FilterMessage" **type**="text" **asp-for**="FilterMessage" />  <input id="CurrentPage" **type**="text" **asp-for**="CurrentPage" />  <input id="PreviousPage" **type**="text" **asp-for**="PreviousPage" />  <input id="SortedColumn" **type**="text" **asp-for**="SortedColumn" />  <input id="FilterType" **type**="text" **asp-for**="SortType" />  </form> |

The above form will be submitted and tells the controller to get the filter data or next page data.

This form is important for filtering, sorting, and pagination.

The detail for the pagination view data is given above in the code structure section. For more reference check out the Companies index.cshtml

## Adding JavaScript Files in Project.

For adding the js files of the related page go to the following path

Path: AgentPortal>wwwroot>js>

Create a folder and file accordingly in the folder and then import it in an index.cshtml file.

## This document ends here