

Cognizant Academy

Vector Web Designs

**ASP.Net Core MVC, Entity
Framework Core,**

**SQLServer - Integrated
Capability Test**

Version 0.1

Table of Contents

1.0 Introduction	3
1.1 Purpose of this document.....	3
1.2 Definitions & Acronyms.....	3
1.3 Project Overview	3
1.4 Use case Diagram	4
1.5 Scope	4
1.6 Target Audience	4
1.7 Hardware and Software Requirement.....	4
1.7.1 Hardware Requirements	4
1.7.2 Software Requirements.....	5
2.0 System Diagram.....	5
3.0 Architecture	6
4.0 Solution Creation.....	6
5.0 Create Division Details	8
5.1 Requirement Flow	8
5.2 Technical Guidelines	10
5.2.1 Implementing POCO/Entity class.....	10
6.0 Create Agent Details	11
6.1 Requirement Flow	11
6.2 Technical Guidelines	14
6.2.1 Implementing POCO/Entity class.....	14
7.0 Stipend Details	15
7.1 Requirement Flow	15
7.2 Technical Guidelines	17
7.2.1 Implementing POCO/Entity class.....	17
8.0 Data Context Implementation	19
9.0 Repository Implementation	19
10.0 Controllers and Views Implementation	20
11.0 Evaluation Areas	22

1.0 Introduction

1.1 Purpose of this document

The purpose of this document is outline the specification for a MVC Core application which uses Entity Framework Core's Code-First approach to connect a database and allows its users to keep a record of web designing.

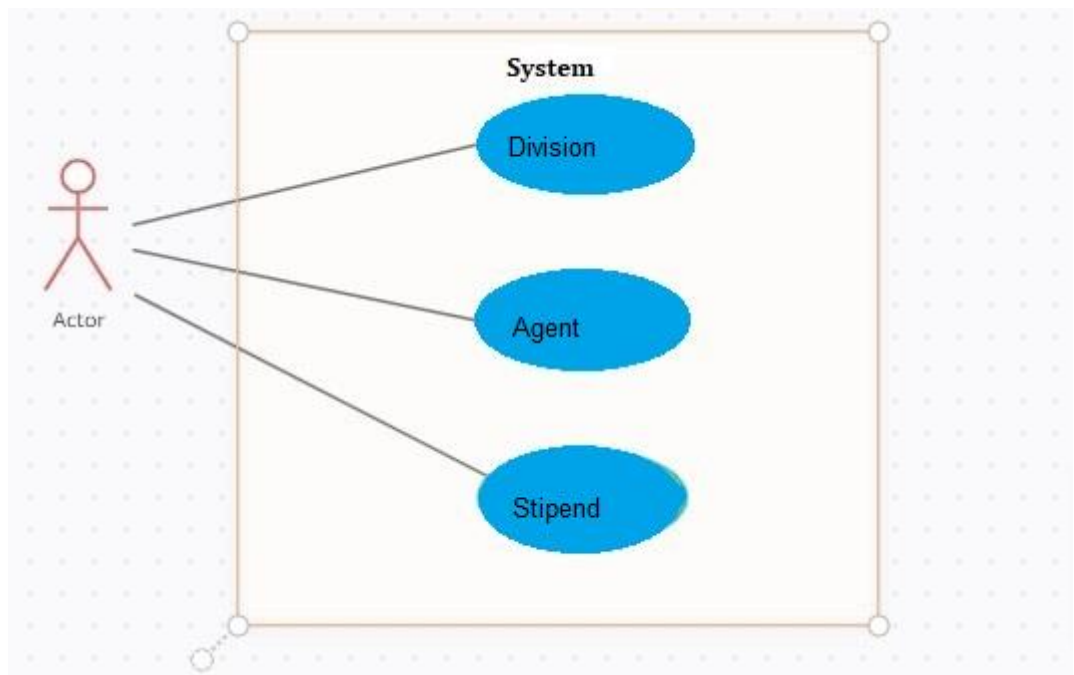
1.2 Definitions & Acronyms

Definition / Acronym	Description
C#	C# (pronounced C sharp) is an object-oriented server-side programming language for developing .NET application
SQL Server	SQL Server is a powerful relational database for storing data
Asp.NET Core MVC	Light weight framework for developing server-side application which provides separation of concerns for developing applications using asp.net core
Entity framework Core	Provides an ORM to map a relational model with the object oriented model.

1.3 Project Overview

Vector Web Design is one of the top most companies in South Asia. This centers are maintaining the records of their Division, Agents and Stipend detail.

1.4 Use case Diagram



1.5 Scope

The Creating an asp.net core mvc application to store and retrieve data from sql server database using entity framework code first approach.

1.6 Target Audience

Advance Level

1.7 Hardware and Software Requirement

1.7.1 Hardware Requirements

#	Item	Specification/Version
1.	PC	8GB RAM

1.7.2 Software Requirements

#	Item	Specification/Version
1.	.NET Framework	6.0
2.	Visual Studio Professional	2022
3	SQLSERVER Enterprise	2014
4	Internet Explorer/Google Chrome/Firefox	-

2.0 System Diagram

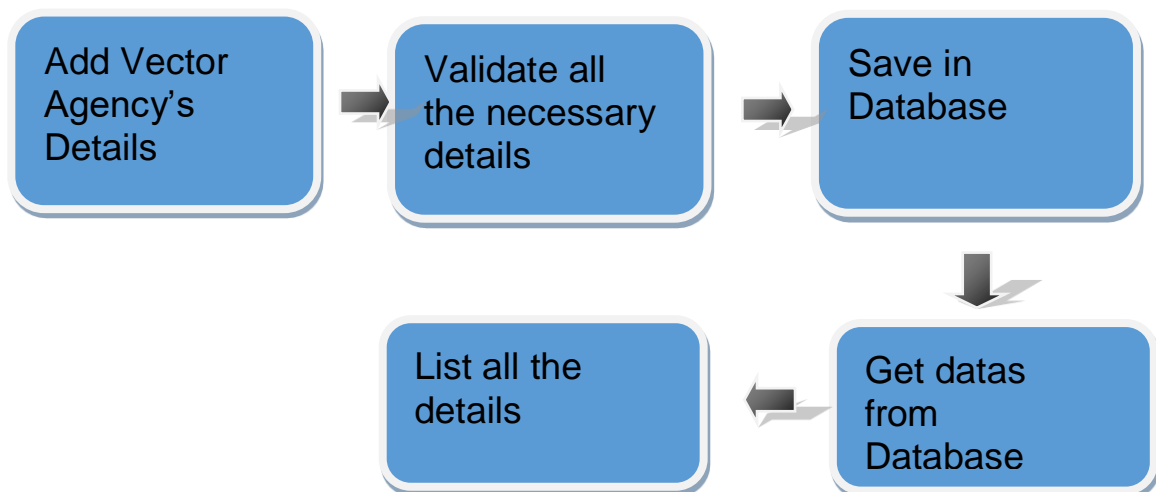


Figure : System Diagram

3.0 Architecture

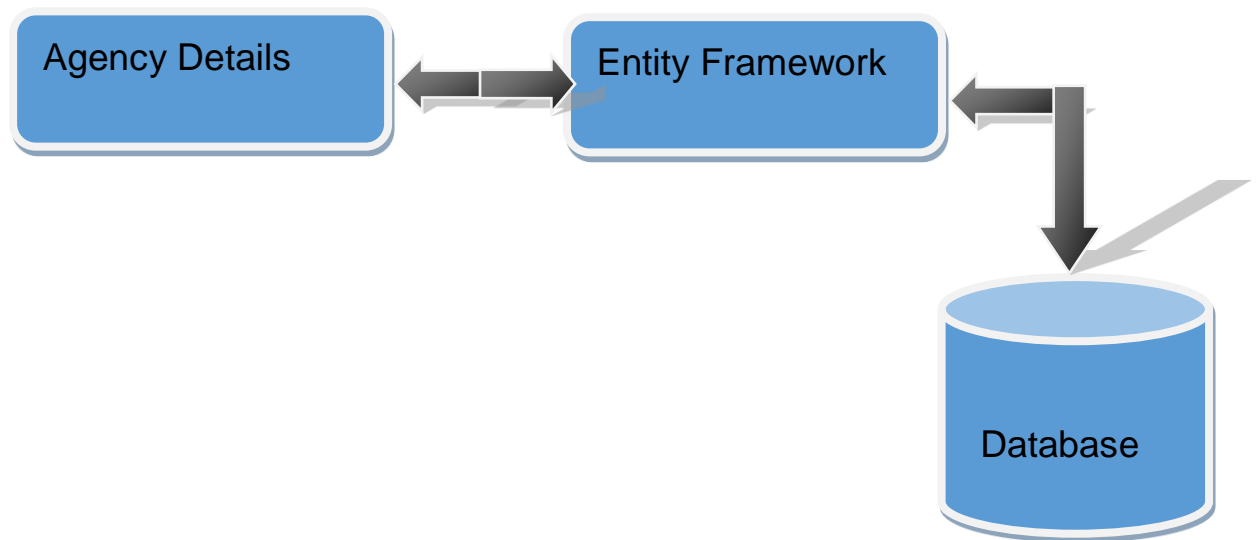


Figure : Architecture Diagram

4.0 Solution Creation

1. Create a new project in visual studio using the ASP.NET Core Web App (Model-View-Controller) template.
2. Give the solution name as **"Vector Web Designs"**
3. Give the project name as **"Vector"**
4. Click the Framework version .NET 6.0.
5. Change the authentication mode to **"None"**
6. Ensure to uncheck the following checkboxes
 1. Docker support
 2. Create unit test for project
7. Delete the default **"VectorController"** and it's associated views folder.
8. Delete the default **"Error.cshtml"** view from **Views/Shared** folder.
9. Click on then Project **"Vector"** and add all the needed nuget packages in that csproj file

```

<TargetFramework>net6.0</TargetFramework>
<Nullable>enable</Nullable>
<ImplicitUsings>enable</ImplicitUsings>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="Microsoft.EntityFrameworkCore" Version="7.0.5" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="7.0.5" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="7.0.5">
    <PrivateAssets>all</PrivateAssets>
    <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
  </PackageReference>
  <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="6.0.16" />
</ItemGroup>
</Project>

```

Figure : Vector.csproj

10. Go to solution explorer, Program.cs - in that set the pattern as per the below snapshot

```

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();

```

Figure : Program.cs Pattern

NOTE :

- A. Solution is already created for you on the platform. Do not make any changes to connection string in **appsettings.json** file on the platform
- B. After creating all 3 models, Follow the below-given instructions to create migrations
- C. For Creating Migration, Use this below code in the Package Manager Console
 1. **add-migration SampleName**- once this migration is successful, then move to next step

2. update-database

You can give any name instead of **SampleName**

5.0 Create Division Details

5.1 Requirement Flow

Steps Explanation :

- 1 User launches the application and index.cshtml page is displayed to the user as follows

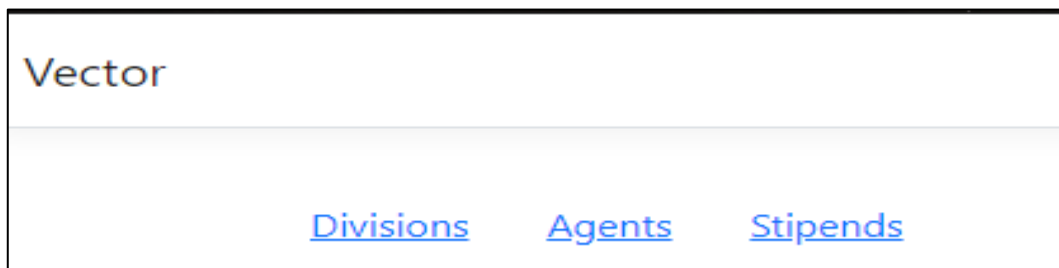


Figure : Index page

- 2 User click Divisions link, the DivisionsList.cshtml page is display to user as follows



Figure : Divisionslist form

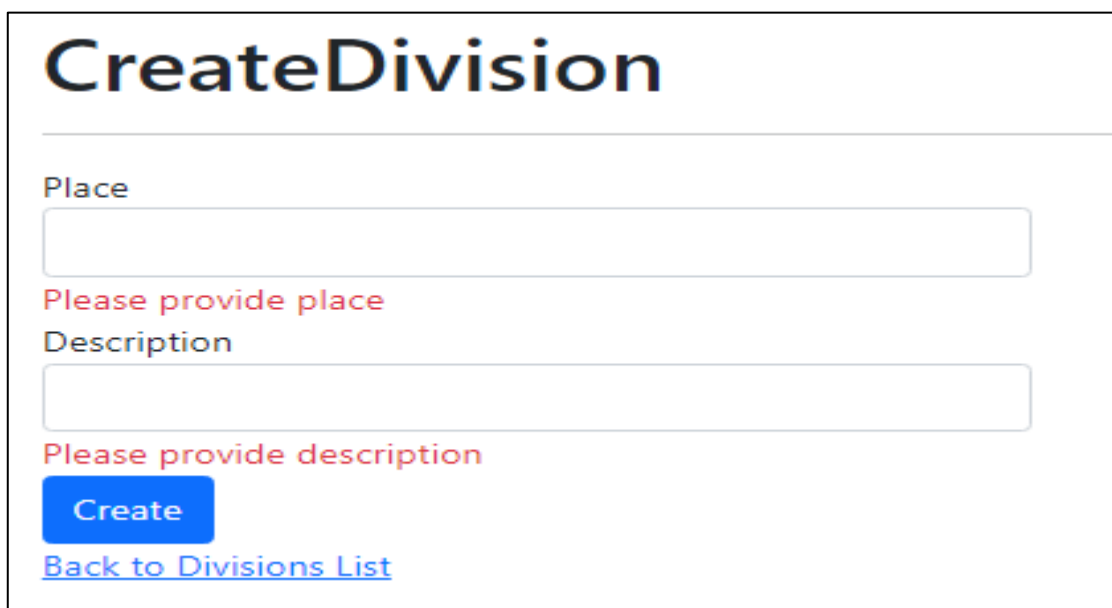
- 3 Then click create new link (CreateDivision.cshtml)



The image shows a web form titled "CreateDivision". It has two input fields: "Place" and "Description". Below the "Description" field is a blue "Create" button. At the bottom of the form is a blue link that says "Back to Divisions List".

Figure : CreateDivision form

- 4 If there is any validation failures application will display appropriate validation message as follows



The image shows the same "CreateDivision" form as in Figure 1, but with validation messages. Below the "Place" input field, there is a red text message: "Please provide place". Below the "Description" input field, there is a red text message: "Please provide description". The "Create" button and the "Back to Divisions List" link are still present at the bottom.

Figure : CreateDivision form validation

- 5 User will fill up all the required details and click the create button which will save the Division details into the database and displays a message to user as follows

CreateDivision

Division Details Added Successfully

Place

Description

[Create](#)

[Back to Divisions List](#)

DivisionsList

[Create New](#)

Place	Description
North Cliff	Generalized Team

Figure : Division Details Added

5.2 Technical Guidelines

5.2.1 Implementing POCO/Entity class

Implementing Division.cs

1. Create a new class in the “Models” folder with the name as “**Division**” with the following specification.

Table : Division class

Property Name	Type	Modifier
Id	int	public
Place	string	public

Description	string	public
-------------	--------	--------

2. Division entity class will be used as POCO class for generating the database table and also for creating strongly-typed views for the actions method.
3. Modify the “**Division**” class with appropriate DataAnnotations to match the following validation rules

Table : Division class validations specifications

Property	Validation	Error Message
Id	Key	
Place	Must not be blank	Please provide place
Description	Must not be blank	Please provide description

6.0 Create Agent Details

6.1 Requirement Flow

Steps Explanation :

1. User launches the application and index.cshtml page is displayed to the user as follows

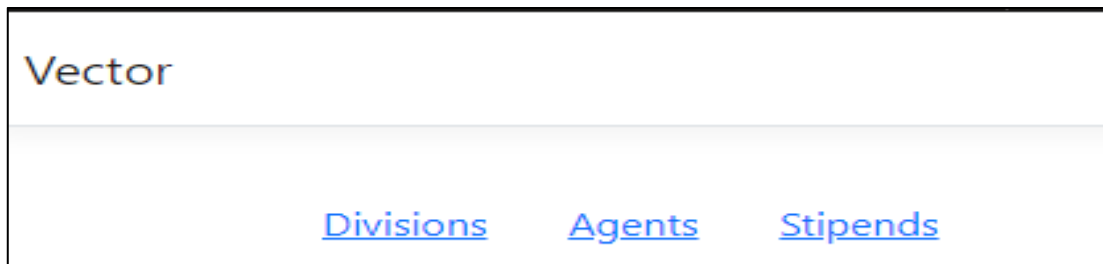


Figure : Index page

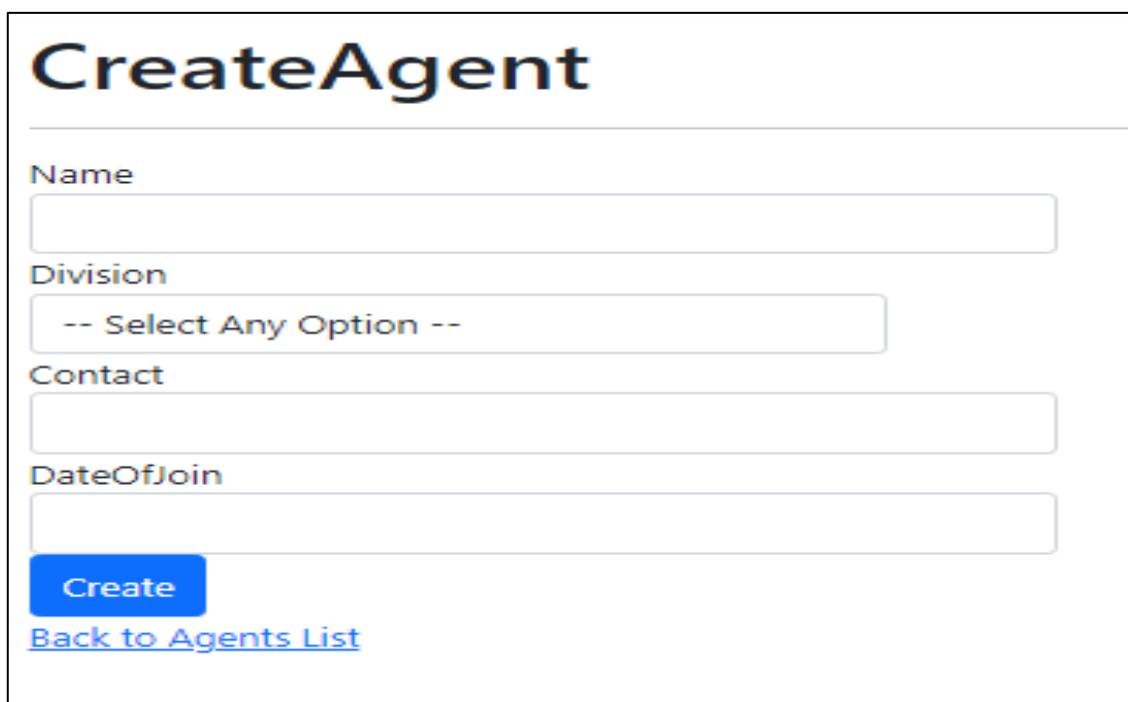
2. User click Agents link, the AgentsList.cshtml page is display to user as follows



The screenshot shows the 'AgentsList' page. At the top left is the title 'AgentsList' in a large, bold, black font. Below it is a blue link labeled 'Create New'. Below the link is a table with four columns: 'Name', 'Place', 'Contact', and 'DateOfJoin'. The table body is currently empty.

Figure : Agentslist form

3. Then click create new link, CreateAgent.cshtml page will display to user as follows



The screenshot shows the 'CreateAgent' page. At the top is the title 'CreateAgent' in a large, bold, black font. Below the title are four input fields: 'Name' (a text box), 'Division' (a dropdown menu showing '-- Select Any Option --'), 'Contact' (a text box), and 'DateOfJoin' (a text box). Below these fields is a blue button labeled 'Create'. At the bottom is a blue link labeled 'Back to Agents List'.

Figure : CreateAgent form

4. If there is any validation failures application will display appropriate validation message as follows

CreateAgent

Name

Please provide name

Division

The value " is invalid.

Contact

Please provide contact

DateOfJoin

Please provide date of join

[Create](#)

[Back to Agents List](#)

Figure : CreateAgent form validation

5. User will fill up all the required details and click the create button which will save the Agent details into the database and displays a message to user as follows

CreateAgent

Agent Details Added Successfully

Name

Division

Contact

DateOfJoin

[Create](#)

[Back to Agents List](#)

AgentsList			
Create New			
Name	Place	Contact	DateOfJoin
Agent 1	North Cliff	9894601290	01/10/2020

Figure : Agent Details Added

6.2 Technical Guidelines

6.2.1 Implementing POCO/Entity class

Implementing Agent.cs

1. Create a new class in the “Models” folder with the name as “**Agent**” with the following specification.

Table : Agent class

Property Name	Type	Modifier
Id	int	public
Name	string	public
DivisionId	int	public
Division	Division?	public
Divisions	SelectList?	public
Contact	string	public
DateOfJoin	string	public

2. Agent entity class will be used as POCO class for generating the database table and also for creating strongly-typed views for the actions method.
3. Modify the “**Agent**” class with appropriate DataAnnotations to match the following validation rules

Table : Agent class validations specifications

Property	Validation	Error Message
Id	Key	
Name	Must not be blank	Please provide name
DivisionId	DisplayName attribute - Division Must not be blank	Please select a division
Division	Foreign Key NotMapped	
Divisions	NotMapped	
Contact	Must not be blank	Please provide contact
DateOfJoin	Must not be blank	Please provide date of join

7.0 Stipend Details

7.1 Requirement Flow

Steps Explanation :

1. User launches the application and index.cshtml page is displayed to the user as follows

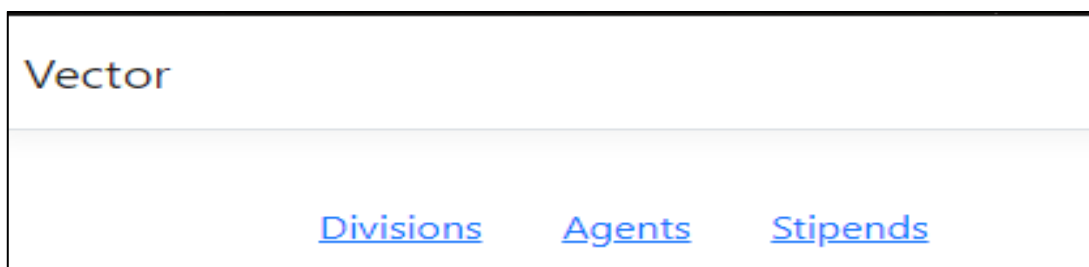


Figure : Index page

2. User click on Stipends link, the StipendsList.cshtml page is display to user as follows

StipendsList		
Create New		
Name	MonthOfStipend	AmountOfSalary

Figure : StipendsList form

- Then click create new link, CreateStipend.cshtml page will display to user as follows

CreateStipend

Agent

-- Select Any Option --

MonthOfStipend

AmountOfSalary

0

Create

[Back to Stipends List](#)

Figure : CreateStipend form

- If there is any validation failures application will display appropriate validation message as follows

CreateStipend

Agent

-- Select Any Option --

The value " is invalid.
MonthOfStipend

Please provide month of stipend
AmountOfSalary

0

Create

[Back to Stipends List](#)

Figure : CreateStipend form validation

5. User will fill up all the required details and click the create button which will save the Stipend details into the database and displays a message to user as follows

CreateStipend

Stipend Details Added Successfully

Agent

MonthOfStipend

AmountOfSalary

[Create](#)

[Back to Stipends List](#)

StipendsList

[Create New](#)

Name	MonthOfStipend	AmountOfSalary
Agent 1	Jan 2021	35701

Figure : Stipend Details Added

7.2 Technical Guidelines

7.2.1 Implementing POCO/Entity class

Implementing Stipend.cs

1. Create a new class in the “Models” folder with the name as “**Stipend**” with the following specification.

Table : Stipend class

Property Name	Type	Modifier
Id	int	public
AgentId	int	public
Agent	Agent?	public
Agents	SelectList?	public
MonthOfStipend	string	public
AmountOfSalary	double	public

2. Stipend entity class will be used as POCO class for generating the database table and also for creating strongly-typed views for the actions method.

3. Modify the “**Stipend**” class with appropriate DataAnnotations to match the following validation rules

Table : Stipend class validations specifications

Property	Validation	Error Message
Id	Key	
AgentId	DisplayName attribute - Agent Must not be blank	Please select an agent
Agent	Foreign Key NotMapped	
Agents	NotMapped	
MonthOfStipend	Must not be blank	Please provide month of stipend
AmountOfSalary	Must not be blank	Please provide amount of salary

8.0 Data Context Implementation

1. Create a new folder and name it as Data and create a new file named “**VectorDBContext**” which inherits from the “**DbContext**” class.
2. Modify the **VectorDBContext** to add following details

Table : FlightContext class

Property Name	Type	Modifier
Divisions	DbSet<Division>	public
Agents	DbSet<Agent>	public
Stipends	DbSet<Stipend>	public

9.0 Repository Implementation

1. Create a new class named “**VectorRepository**” in the Repository folder with the following specification

Table : VectorRepository class

Field Name	Type	Modifier
_context	VectorDBContext	private

Table : VectorRepository Constructors

Constructor Type	Input Parameters	Modifier
Default	-	public
Parameterized	VectorDBContext	public

Table : VectorRepository class methods

Method Name	Input Parameters	Return Type	Modifier
AddDivision()	Division	bool	public
AddAgent()	Agent	bool	public
AddStipend()	Stipend	bool	public

2. Use the constructor to create a new instance of the **DbContext** class.
3. Implement the **AddDivision()**, **AddAgent()** and **AddStipend()** to save details into the database.
4. Based on whether details are saved or not method should return true/false.

10.0 Controllers and Views Implementation

1. Add a new controller named **"HomeController"** in the controllers folder with the following specification.

Table : HomeController Actions

Action Name	Input Parameters	Http Request Type	Modifier	Return Type
Index()	-	Get	public	ActionResult

2. Go to Index.cshtml set title as **"Index"**.
 - a. Go to Index.cshtml view and add a hyperlink to Vector Controller "DivisionsList" action with id="lnkDivisions" attribute.
 - b. Go to Index.cshtml view and add a hyperlink to Vector Controller "AgentsList" action with id="lnkAgents" attribute.
 - c. Go to Index.cshtml view and add a hyperlink to Vector Controller "StipendsList" action with id="lnkStipends" attribute.
3. Once user click on the above link, it should show the appropriate list in the page. All 3 lists page should add a link button for **create new** - it should navigate to create pages
4. Add a new controller named **"VectorController"** in the controllers folder with the following specification.

Table : VectorController Fields

Field Name	Type	Modifier
_repository	VectorRepository	private
_sale	VectorDBContext	private

Table : VectorController Constructors

Constructor Type	Input Parameters	Modifier
Parameterized	VectorDBContext	public

Table : VectorController Actions

Action Name	Input Parameter S	Http Request Type	Modifie r	Return Type
DivisionsList()	-	Get	public	ActionResul t
AgentsList()	-	Get	public	ActionResul t
StipendsList()	-	Get	public	ActionResul t
CreateDivision()	Division	Post ValidateAntiForgeryToke n	public	ActionResul t
CreateAgent()	Agent	Post ValidateAntiForgeryToke n	public	ActionResul t
CreateStipend()	Stipend	Post ValidateAntiForgeryToke n	public	ActionResul t

5. Implement the constructor to instantiate the repository instance.
6. Implement the CreateDivision(), CreateAgent() and CreateStipend() action methods with [HttpGet].
 - a. Use Scaffold the view using Create template.
 - b. For CreateDivision - Set an ID attribute on the submit button with the value "**btnCrDivision**"
 - c. For CreateAgent - Set an ID attribute on the submit button with the value "**btnCrAgent**"
 - d. For CreateStipend - Set an ID attribute on the submit button with the value "**btnCrStipend**"
7. Implement the CreateDivision(), CreateAgent() and CreateStipend() action for http postrequest to carryout the following operations
 - a. For all 3 post actions Validate the model and return the view if

- model is invalid
 - b. If model is valid save the model in the database using the repository object's method.
 - c. When the Division details are saved successfully in the database, the put an acknowledgement in ViewBag by creating a **"Message"** property with value as **"Division Details Added Successfully"**
 - d. When the Agent details are saved successfully in the database, the put an acknowledgement in ViewBag by creating a **"Message"** property with value as **"Agent Details Added Successfully"**
 - e. When the Stipend details are saved successfully in the database, the put an acknowledgement in ViewBag by creating a **"Message"** property with value as **"Stipend Details Added Successfully"**
8. Modify the Create views to display the value of ViewBag's Message property inside an <h2> element. Assign the ID=**"Message"** attribute to <h2>
 9. For all the Lists page, the table tag should have the same id as per the below-given id

DivisionsList Table Id - **tblDivisions**

AgentsList Table Id - **tblAgents**

StipendsList Table Id - **tblStipends**

11.0 Evaluation Areas

01	Launch of the application from Division, Agent and Stipend pages
02	Logic in create functionality and Success message
03	Validation of input on controls on all pages
04	Creating properties and get post method checking
05	Checking ability in razor engine