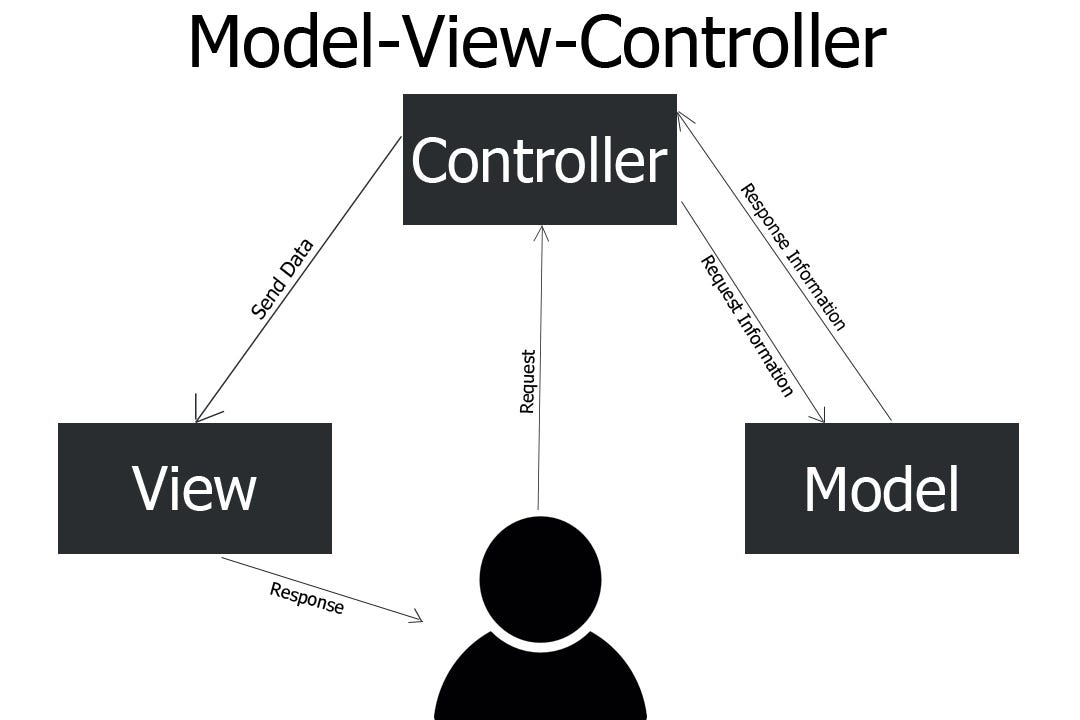
**Day1(MVC)**

**REF: Day25**

MVC (Model-view-controller)

* It is the web development framework.
* MVC is a design pattern used to decouple user-interface (view), data (model), and application logic (controller).



1. A controller is responsible for controlling the way that a user interacts with an MVC application.
2. My output is my view.
3. Controller is the function and our console.write is similiar to our view in MVC.
4. function get the data from using entity framework from database and it is shown to the user using View.
5. passing data from Function(Controller) to view --> ViewBag, viewData, Tuple e.t.c

5.1. techniques to pass data from the controller to view.

5.2 To maintain **Single Responsibility Principle (SRP)**

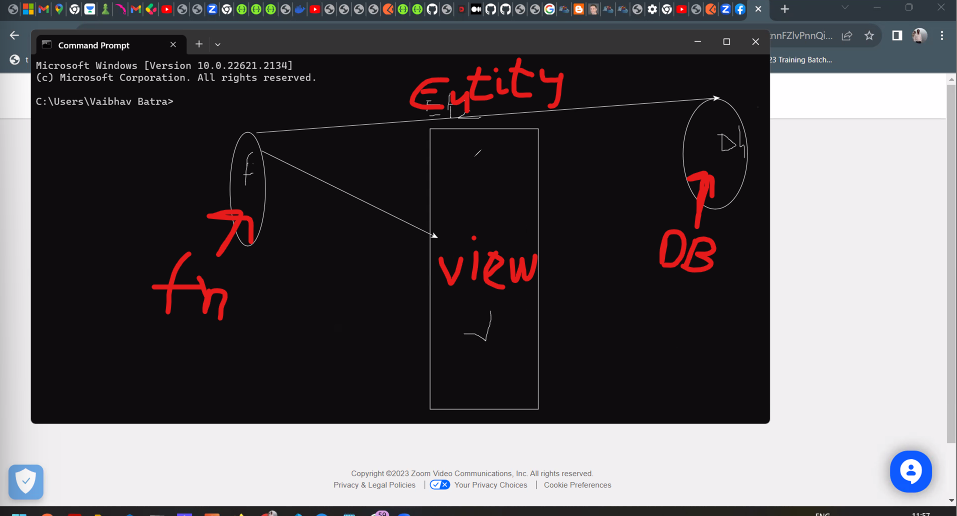
5.2.1. The idea behind the SRP is that every class, module, or function in a program should have one responsibility/purpose in a program.

6. **Class is called as controller** and **function is known as Action** in MVC.

7. Controller is a class and user will request to controller and not function.

8. On receiving request from the user class is called and in background an object of the class is made which will called the required function.

* In mvc the extension of **view is .cshtml**
* **why .cshtml and not .html?** because we can do programming in both function and view and we can add server-side-logic(c#) in html.
* \*\* If we do programming in View then it is known as **Razor Programming.**
* Bootstrap and J.Query are applied in **View**.

****

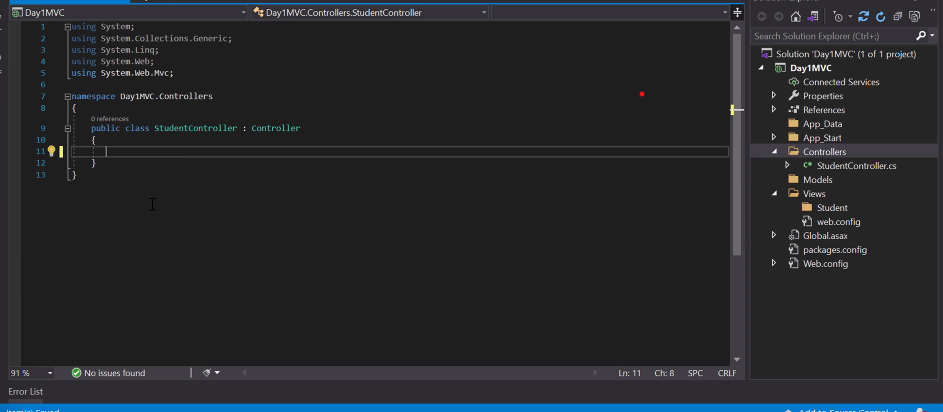
* Function get the data from database using entity framework and store it in generics/collection and then send it to view.

**Q1. what is the use of RouteConfig.cs ?**

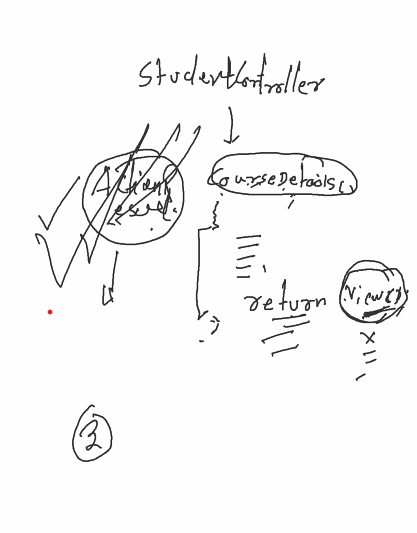
**Q2. What is the use of Global.asax file in MVC ?**

**Q3. what is the use of web.config file in MVC?**

* **controller -->right-click --> add controller(top) --> MVC 5 Controller -empty --> controller name: student**

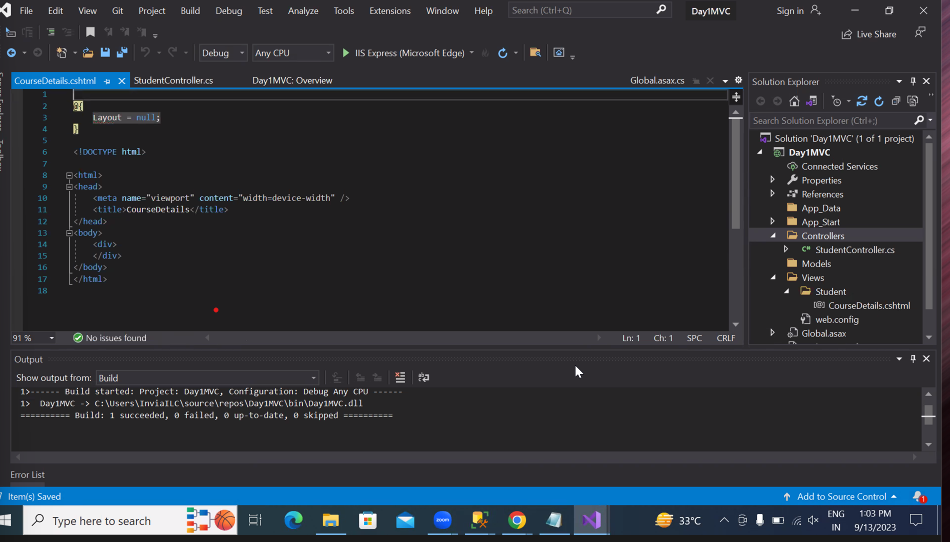
****

If we want to return view from the function then its return type is **ActionResult** and return value is **View().**

****

**Actionresult** is class and it is the return type of the function when we want to link function with view.

right click on function --> Add view --> **take default view name(view name is similiar to function).**

****

the newely created view is stored in view folder.

There are Multiple ways to pass data from Action(Function) to View.

1. ViewBag,ViewData, TempData,ViewModel,Tuple, etc \*\*-- recommend: viewModel --\*\*

**viewbag**

* It is a predefined object which is used to pass data from function to view.
* after that scope of ViewBag is finished.
* One-way functioning(one viewBag object per variable).

**URL -> Domain\className\function**

* Course co = ViewBag.Temp as Course; --> as used for explicit conversion
* is is a operator to check type of the variable.

we are telling the compiler that the Temp is of object type and we are converting it to Course type **so that implicit conversion by compiler will not degrade our App performance.**

**----------------------------------------------------**

**Day2(MVC)**

**REF: Day26**

* So in last session, we discuss the concept of how to pass data from action to view in multiple ways.
* Strongly recommended --> ViewModel()
* There are multiple ways we can make UI known as form programming.

1. Request Object

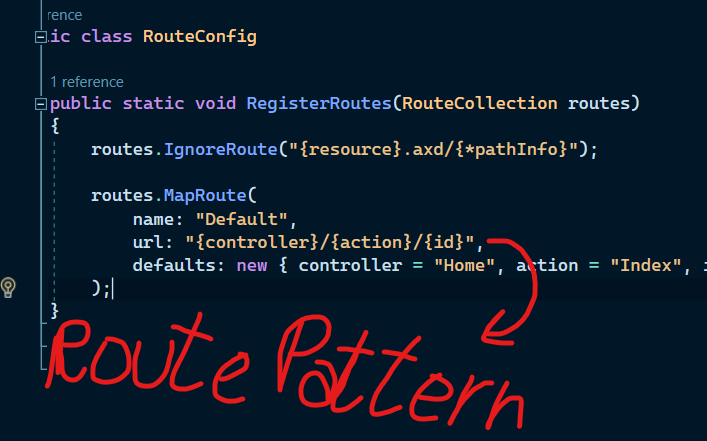
2. FormsCollection

3. ViewModel \*\* Recommended

4. Ajax(Asyncronous javascript xml) Jquery

Q .Who create route table in MVC?

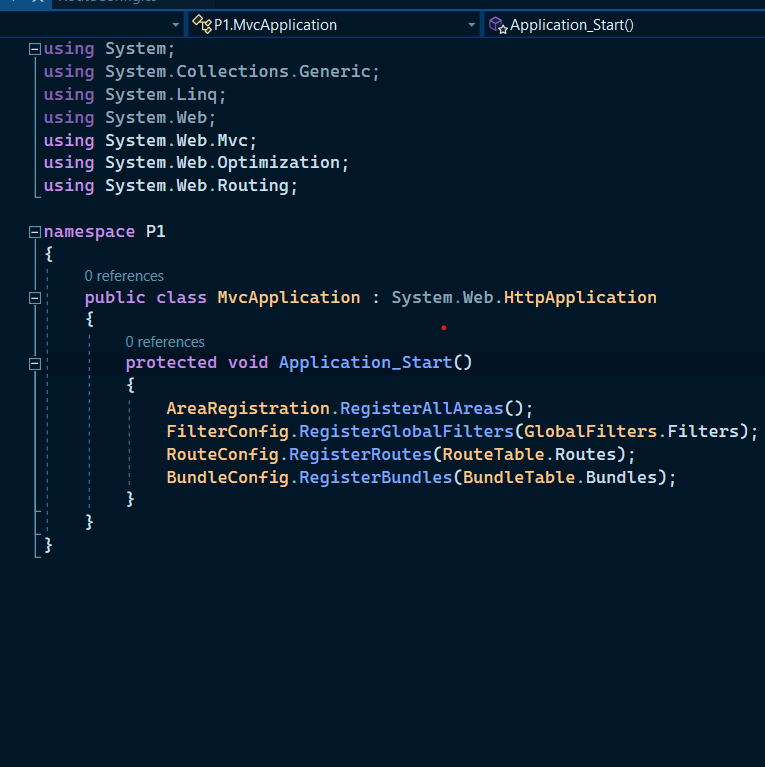
**Ans. RegisterRoutes** Function of the Route Config class is used to create route table in MVC.



* defaults:--> decide which action of which route should run first everytime we run our MVC Application.
* In route table we can map multiple routes but **it is mandatory to add default route it define the route of our first page in our website(LANDING PAGE).**
* There should be only one Map Function.
* viewbag is the extended version of ViewData as viewdata store data in object (boxing and unboxing).

**Architecture of MVC**

* Program start from the **Global.asax** file and in there we have a class called **MvcApplication and** it has a static event(method) **Application\_start()** from there we can check the flow of program,first the area is defined then the filter and then the routes.



* Now in RouteConfig.cs file we define the routes where we need the user request to redirect basically routing is a pattern matching algorithm.

**----------------------------------------------------**

**Day3(MVC)**

**REF: Day27**

**Mvc Request Lifecycle**

* Asp.net core work on **Kestrel Web Server based on Libuv library** (same used by Node.js) Libuv supports an event-driven style of programming. Some of its core utilities include:

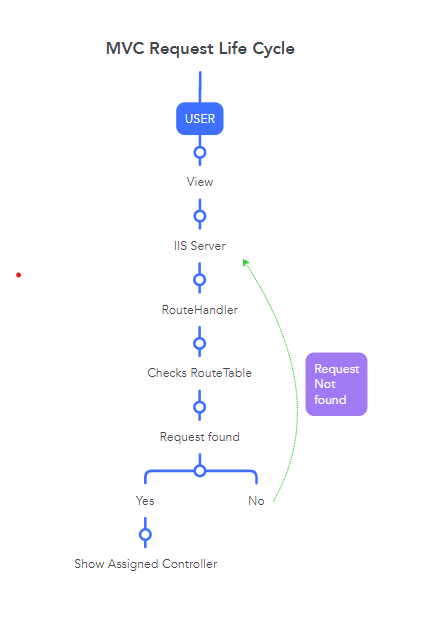
1. Non-blocking network support

2.Asynchronous file system access

3. Timers

4. Child processes

* Applications are often written to respond to human actions. With event-driven programming, there is a loop that listens for events. It then triggers a callback function. To reduce the number of SYS calls, all other work is executed in managed code on standard .NET worker threads.
* **Routerhandler(class)** will lookup for the requested url in **routeTable** if found then it forward the assigned controller otherwise it send the request to the Server(**IIS**)**.**
* IIS has custom exception handler to handle "URL not found".
* **\*RouterHandler is a predefined class which checks for request url in RouteTable.**

****

Q. Is there a MAIN() Function in MVC? How memory of object are allocated without a main() in MVC?

Ans. Yes there is Main(), **DefaultControllerFactory** This is the main method which is used for creation and dynamic loading of the controller.

Whenever the request comes from the users, it is processed in the following sequence in order to render the view to the user.



----------------------------------------------------

----------------------------------------------------

**Day3(MVC)**

**REF: Day29 ( Day 28th class is for practice only)**

* Attributes are extra behaviour of an object(Annotations)(Extra-addons).
* Attribute enhance the appearence of an object.
* Written as **'[ Attribute\_Name]'.**
* There are three ways of Using Entity framework.
* Code first Approach, Database first Approach and Model first approach.
* In Code First Approach **The tables are created on runtime and not before.**
* **Migration is a technique which is applied mainly to convert Class attributes into database properties.**
* In Database first approach we use **Scaffolding** to convert database tables to classes.
* **Scaffolding :-** ASP.NET Scaffolding is a code generation framework for ASP.NET Web applications. Visual Studio 2013 includes pre-installed code generators for MVC and Web API projects.
* **\*\*** Code first Apprach(CFA) is **Early Binding** and Database-first Approach(DFA) is **Late Binding**.
* If the Tables are more then database first approach is preferred and if tables are small in number then CFA is preferred.
* There are Two ORM(Object relation Mapper) in MVC5: **Entity Framework** and **Dapper(use sql queries instead of LINQ so it has only DFA).**
* The Entity Framework Package has two classes **DbContext** and **DbSelect.**
* **DbContext has constructor** chaining so it helps to perform DML Operation in database.
* There is only one DbContext in a database as it help to connect to Database.
* There is only **one DbSet per table**. Multiple DbSet in a database.
* DbSet is a **Generic class** so all the reterival work is done by this class,searching the table, joins the table.
* To implement multiple operations on a dabase table we can make a interface and add all the functions to be performed in it.
* Make a folder **Functionality** Then define **IEmployeeRepository.cs Interface** and add all the functionality in it.
* Make a class **EmployeeRepository.cs** which implement that interface and this class will be inherited by all the controller we need to perform **C.R.U.D** operations on the Table.
* class **EmployeeRepository.cs** will call the functions in DBContext and DbSet which on our behalf will perform all the operations defined in the class **EmployeeRepository.cs.**
* Now all the logic is defined and it doesen't matter what Frontend framework you will use View() will change and all the other things(controller,action,repository) will remain the same.

---I**EmployeeRepository.cs** ----

using System;

using System.Collections.Generic;

using WebApplication1.Models;

namespace WebApplication1.Repository

{

public interface IEmployeeRepository

{

// to get all the records

IEnumerable<Employee> GetEmployees();

// to search for a single record

Employee SearchEmployees(int id);

// to add new employee record

int CreateEmployee(Employee emp);

// to UPDATE employee record

int UpdateEmployee(Employee emp);

// to DELETE employee record

int DeleteEmployee(int id);

}

}

-- **EmployeeRepository.cs** --

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using WebApplication1.Models;

namespace WebApplication1.Repository

{

public class EmployeeRepository : IEmployeeRepository

{

int IEmployeeRepository.CreateEmployee(Employee emp)

{

throw new NotImplementedException();

}

int IEmployeeRepository.DeleteEmployee(int id)

{

throw new NotImplementedException();

}

IEnumerable<Employee> IEmployeeRepository.GetEmployees()

{

throw new NotImplementedException();

}

Employee IEmployeeRepository.SearchEmployees(int id)

{

throw new NotImplementedException();

}

int IEmployeeRepository.UpdateEmployee(Employee emp)

{

throw new NotImplementedException();

}

}

}

------------------------------------------------------------------------------

* Make a Folder Database in the Root direcotry and make a class **EmployeeManagmentContext.cs** Which will implement Entity Framework and call the database.
* We are using **Context** as suffix becuase this tell that we are using Entity Framework here.
* **\*\* Root namespace of Entity Framework:- System.Data.Entity.**
* In this **System.Data.Entity** namespace we have DBContext and DbSet class.
* \*\* Most of the Functions in DbContext class are **Virtual** and also support **Constructor Chaining**(to pass data to the paremeterized constructor of the parent class usign the constructor of the child) so we need to inherit and overide the fucntions and pass data to the parent using chaining.
* \*\* **variable and Object of a class should not be public.**
* Add entity connection string in **Web.config** below <configSections>

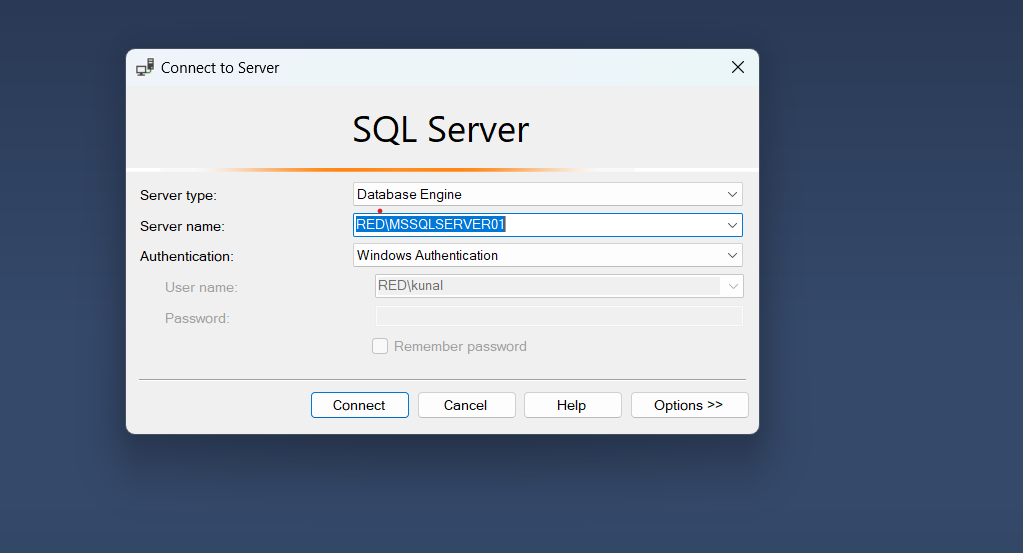
<connectionStrings>

<add name="MyConnection" connectionString="Data Source = RED\MSSQLSERVER01;

Initial Catalog=EmployeeDb; Integrated Security=True" providerName="System.Data.SqlClient" />

</connectionStrings>

* Connection string is the **server name** we get in sqlserver.



**EmployeeManagmentContext.cs**

using System;

using System.Collections.Generic;

using System.Data.Entity;

using WebApplication1.Models;

namespace WebApplication1.Database

{

public class EmployeeManagmentContext : DbContext

{

/\* using constructor chaining to call the parameterized

constructor of the base class and passing our database

connection string \*/

public EmployeeManagmentContext() : base("MyConnection")

{

/\* employees is an object of the DBSet class

In code-first-approach the table name is the name

of the object of the DB Context and column name

are the 'Employee' variables defined in model \*/

}

public DbSet<Employee> employees { get; set; }

}

}

* For **Migration** we need three things: Model, web.config, DbSet.
* Commands to add Migrations in MVC5.

enable-migrations

add-migrations

**Day4(MVC)**

**REF: Day30**

# Flow of Control in code-First-Approach (CFA)

* + First the user enter the data in the form created on view then it send the data on controller then the Controller create call the interface and we create a object of interface with memory allocation of class in repository and then it cal the Context class in the database which will eventually perform DML operations on the table in our database.
  + **IEmployeeRepository obj = new EmployeeRepository();**

public class EmployeeRepository : IEmployeeRepository

{

private EmployeeManagmentContext db = new EmployeeManagmentContext();

int IEmployeeRepository.CreateEmployee(Employee emp)

{

db.employees.Add(emp); // it will add new record in local memory

db.SaveChanges(); // will commit to database

}

}

* + Here we create a object of the Context class in repository