CSE3200 : Software Development - V

Introduction to ASP.NET MVC
LAB - 3

Outline

Introduction to MVC

- ASP.NET is a web application framework from Microsoft
- It is open source
- Applies the general Model-View-Controller Pattern
- Separates the data access logic from display logic
- Popular MVC Frameworks: ASP.NET MVC, Ruby on Rails, Express
- MVC has 3 main aspects:
    - **Model**
    - **View**
    - **Controller**

Understanding MVC Pattern

- **Models -** A set of classes that describes the data you are working with
  - Domain Model
  - View Model

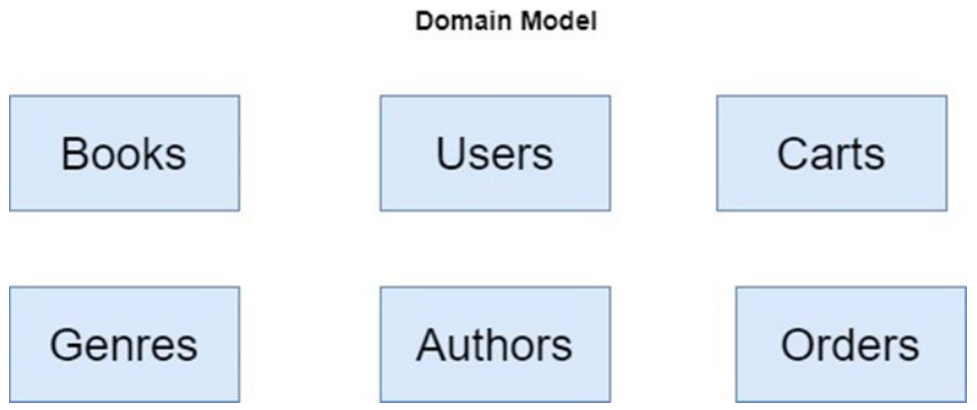
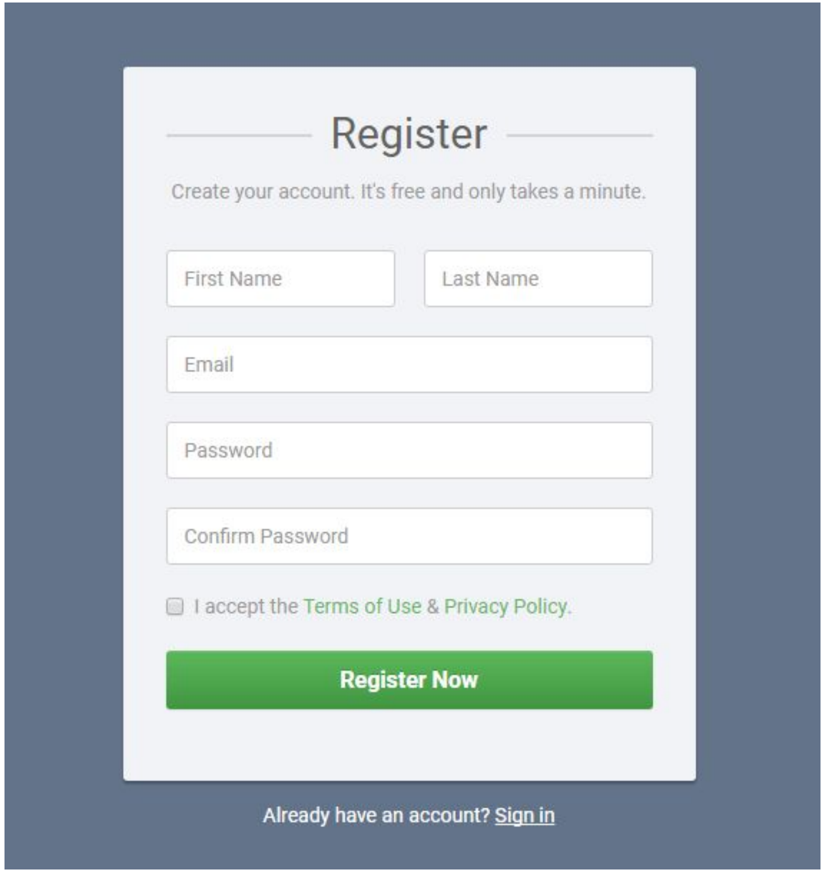


Fig: Example of a Domain Model

Understanding MVC Pattern

- **View -** Defines how the application's UI will be displayed
- The HTML Markup that we display to the user
- Reads data from Model


Register

Create your account. It's free and only takes a minute.

First Name

Last Name

Email

Password

Confirm Password

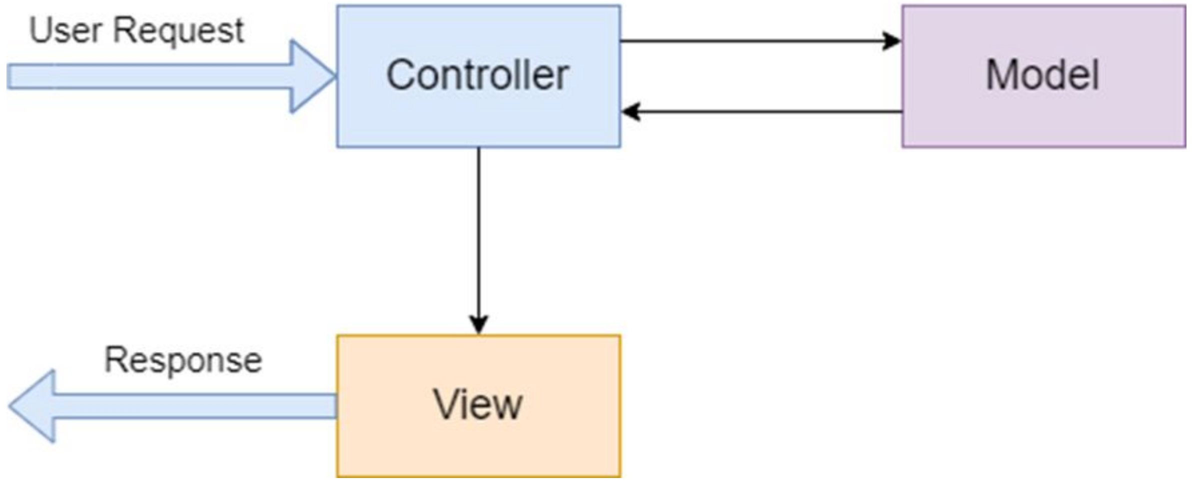
I accept the Terms of Use & Privacy Policy.

Register Now

Already have an account? Sign in

Understanding MVC Pattern

- **Controllers** - a set of classes that receives user request, fetch suitable resources for the task and select proper view to respond back to user
- Controller receives request from browser, call the model, call the view



Folder Structure of MVC

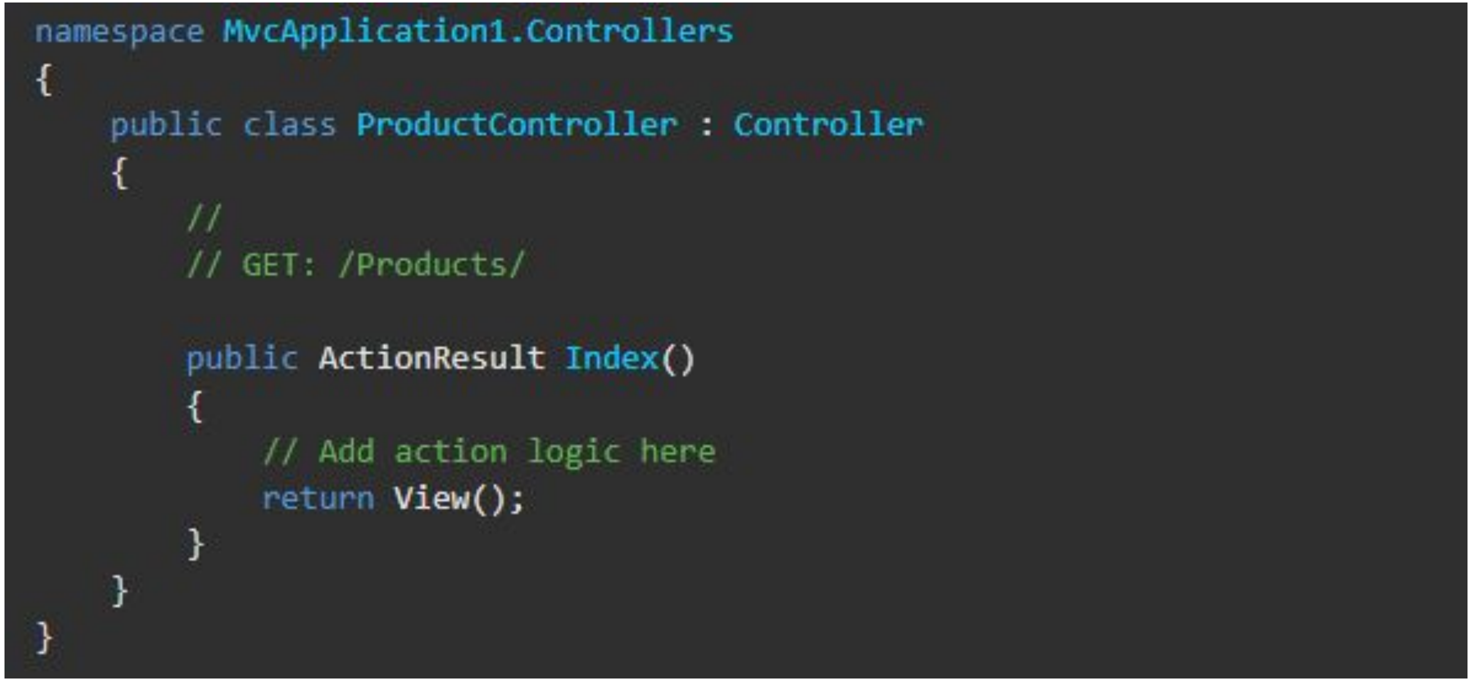
PorjectFolder:

- **\App_Start** - Contains the files that needs to be executed on the first request
- **App_Data** - Contains SQL Server Local DB database files
- **\Controllers** - Contains all controller classes
- **\Models** - Contains all model classes
- **\Views** - Contains all views
- **\Views\web.config** - Contains configuration settings for all views
- **\Global.asax** - Contains application level and session level events
- **\packages.config** - Contains list of NuGet packages currently installed in the project
- **\Web.config** - Contains web application configuration settings, that needs to be initiallized on each request

Controller

- Controller is a class
- Optionally, it’s a public class
- Controller should be inherited from “System.Web.MVC.Controller” class
- Controller’s name should have suffix “Controller”. Ex - ProductController
- All the methods in controller class are by default **Action Method**
- It is common to write the return type of Action Methods as ActionResult

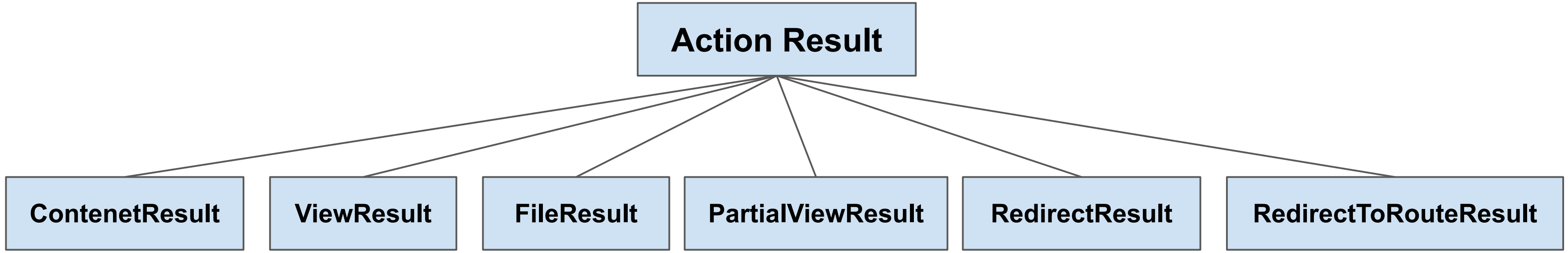
Controller



```
namespace MvcApplication1.Controllers
{
    public class ProductController : Controller
    {
        //
        // GET: /Products/

        public ActionResult Index()
        {
            // Add action logic here
            return View();
        }
    }
}
```

ActionResult

- ActionResult is a class that represents **“result of action method”**
- It is recommended to define action methods return type as **“ActionResult”**
- ActionResult is an abstract class that has several child classes



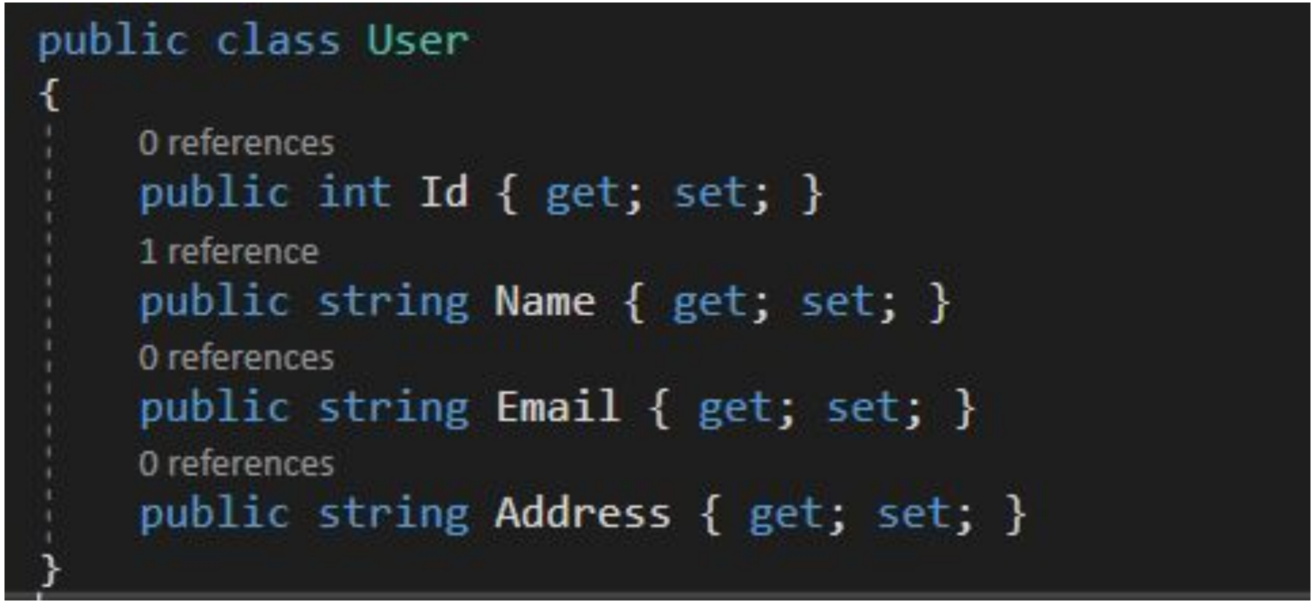
Methods of different types of Action Result

ContentResult	Content(string Content, string ContentType)
ViewResult	View(string ViewName)
FileResult	File(string FilePath, string ContentType)
JsonResult	Json(object data, JsonRequestBehavior behavior)
RedirectResult	Redirect(string url)
RedirectToRouteResult	RedirectToAction(string ActionName, string Controllername)
PartialViewResult	PartialView(string ViewName)

Models

- Model is a class that defines structure of the data that you want to store/display
- It contains business logic (e.g. validation)
- Model will be called by Controller and View
- Domain Model: Represents the structure of the data you want to store in database table (e.g. user information)
- View Model: Represents the structure of the data you want to display to user (e.g. login page)

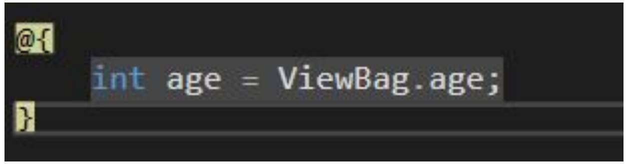
Model


```
public class User
{
    0 references
    public int Id { get; set; }
    1 reference
    public string Name { get; set; }
    0 references
    public string Email { get; set; }
    0 references
    public string Address { get; set; }
}
```

View & Razor View

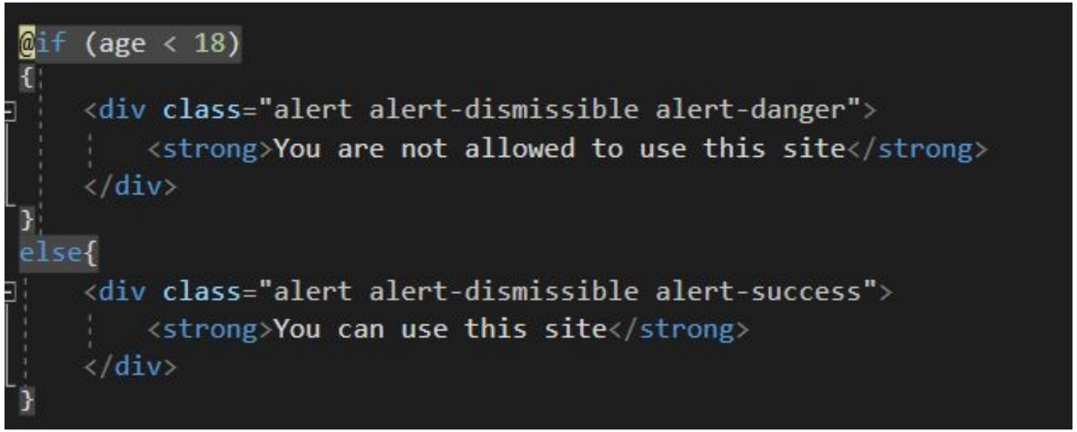
- View is a combination of HTML and C# code
- C# code written within @{} symbol
- Razor View Engine provides set of syntaxes to write C# code in view
- Razor View Engine is responsible to render the view as html
- File extension is .cshtml (.vbhtml)

Razor Syntax



```
@{
    int age = ViewBag.age;
}
```

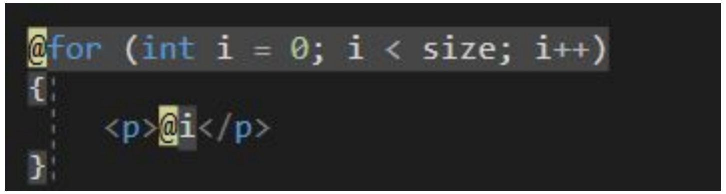
Fig: Razor Block



```
@if (age < 18)
{
    <div class="alert alert-dismissible alert-danger">
        <strong>You are not allowed to use this site</strong>
    </div>
}
else{
    <div class="alert alert-dismissible alert-success">
        <strong>You can use this site</strong>
    </div>
}
```

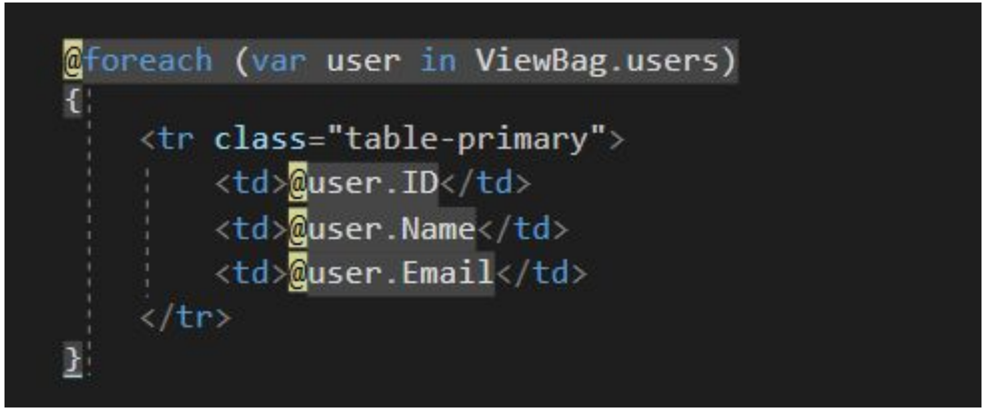
Fig: Razor if-else

Razor Syntex



```
@for (int i = 0; i < size; i++)
{
    <p>@i</p>
}
```

Fig: Razor for



```
@foreach (var user in ViewBag.users)
{
    <tr class="table-primary">
        <td>@user.ID</td>
        <td>@user.Name</td>
        <td>@user.Email</td>
    </tr>
}
```

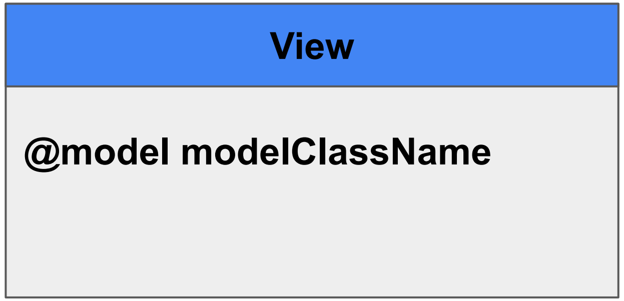
Fig: Razor foreach

Passing Data From Controller to View

- ViewBag
- ViewData
- model

Strongly Typed Views

- Views that is associated with specific type of model class is called strongly typed view
- Strongly typed views have to specify the model class name with @model derivative at the top of the view
- Strongly type view can receive object of that model from the controller

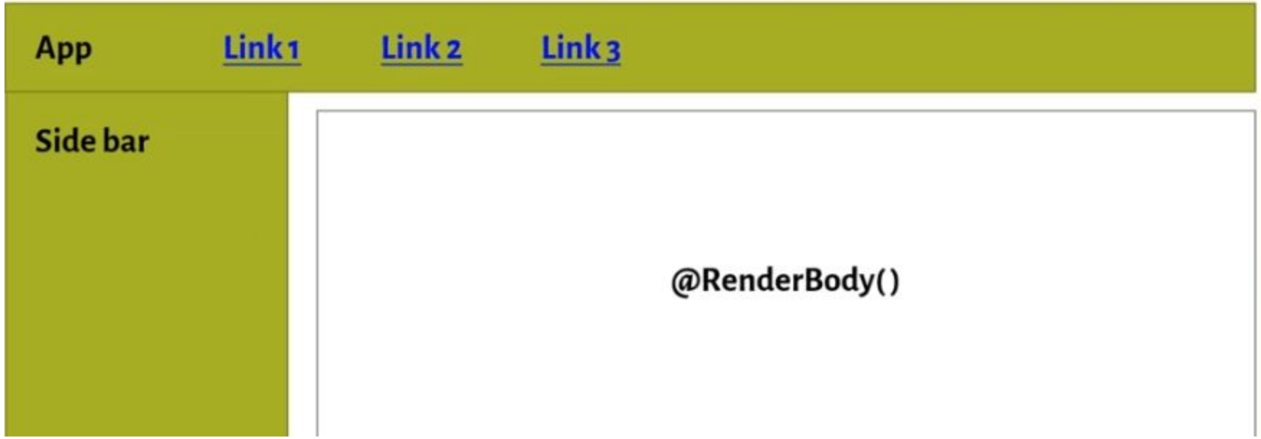


Shared Views

- Shared views are present in the “Views\Shared” folder
- Shared views can be called from any controller
- The views that belong to multiple controllers are created as shared views
- It first searches the view in “Views/ControllerName” folder. If no view is found it searches in the “Views/Shared” folder

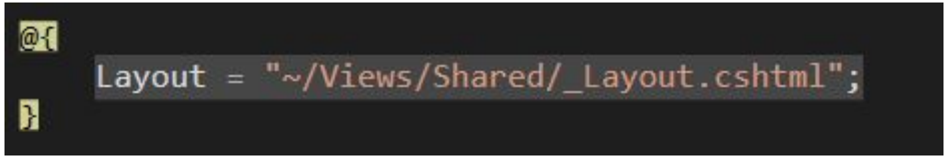
Layout Views

- Layout views contains the common parts of UI. Such as logo, haeder, footer, menubar, sidebar etc.
- **@RenderBody()** method represents the reserved area for the actual content of the view



Layout Views

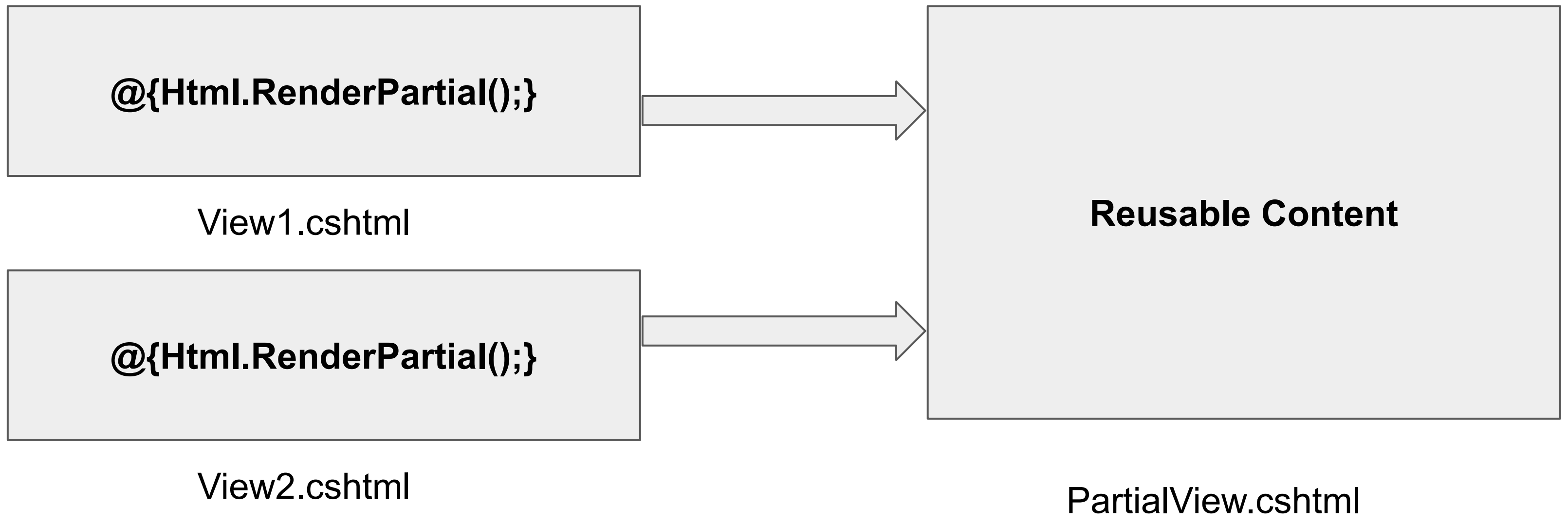
- Data can be shared from a normal view to layout view using Viewbag
- **_ViewStart.cshtml** in Views folder defines the default layout view of all the views of a folder
- There can be multiple layout views in a project (e.g. one layout for user section and one section for admin section)



```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

Partial Views

- Partial view is a small view that contains the content that can be shared among multiple views



URL Routing

- URL Routing is a pattern matching system that monitors the incoming request URL and figure out what to do with that
- It allows you to create the meaningful URLs, instead of mapping to physical files
- Route is a URL pattern which includes literals/parameters
- Literal is fixed, whereas parameter is variable
- Ex - Users/Details/{userid}

/Users/Index

/Users/Contact

/Users/Details/1

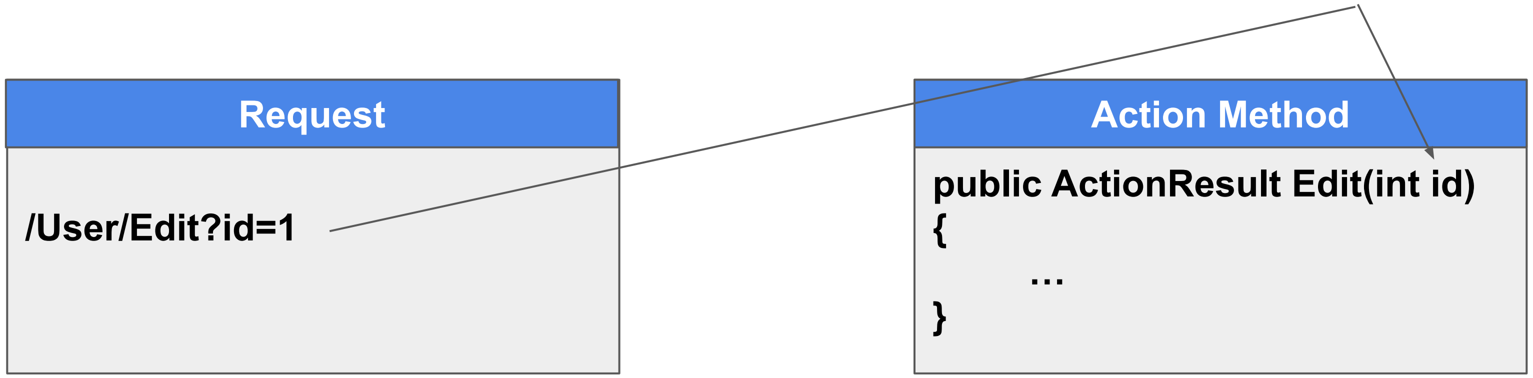
Attribute Routing

- Conventional Routing is difficult for developers to understand which route for which action methods
- Some routes for multiple action methods, some for other. Overall it looks cumbersome
- To overcome this, Attribute Routing is introduced in MVC5
- Attribute routing should be enabled using route.MapMvcAttributesRoutes() in RouteConfig.cs

```
[“url”]

Public ActionResult MethodName()
{

}
```

Model Binding

- Process of receiving values from different sources of the request and passing them as arguments to action method
- Assigns values to different parameters of the action method automatically

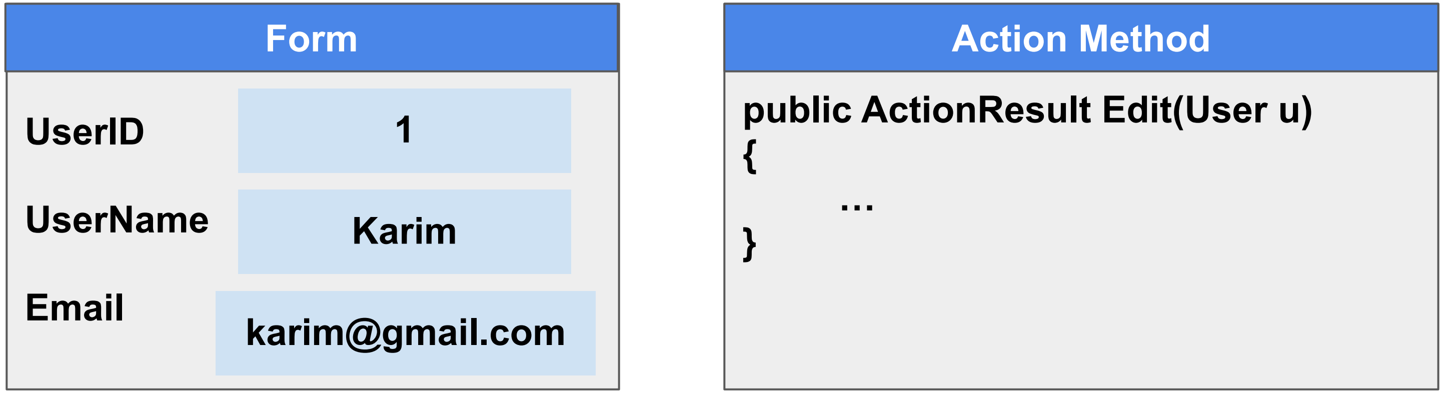


```
/User/Edit?id=1
```

```
public ActionResult Edit(int id)
{
        …
}
```

Model Binding

- Model Binding can work with complex types
- Model Binding can automatically convert form field data or query string values to the properties of complex type parameter of an action method
- If no data passed, default values will be assigned (null or 0)



Form	
UserID	1
UserName	Karim
Email	karim@gmail.com

Action Method

```
public ActionResult Edit(User u)
{
        …
}
```

Model Binding

Common Sources of Model Binding

- Query String
- Form Data (Ex: <input type=”text” name=”UserName”>)
- Route Data (Data passed from other action methods while redirecting)
- JSON request body

Model Binding: Bind Attribute

- [Bind] Attribute allows you to specify the list of properties that you want to bind into the model object
- It allows you to specify “include and exclude”

  comma-separated list of properties

Model Class

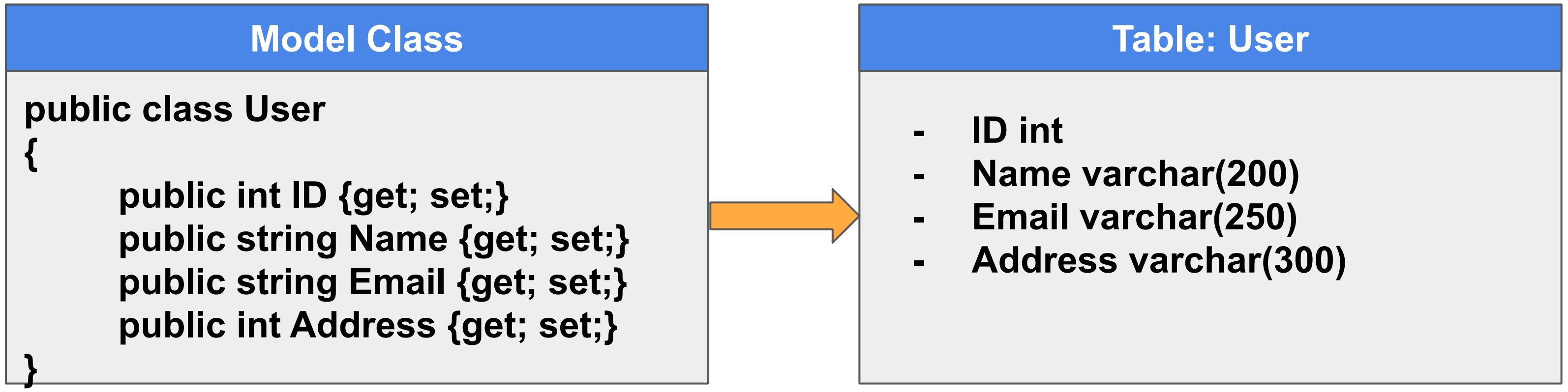
```
public class User
{
        public int ID {get; set;}
        public string Name {get; set;}
        public string Email {get; set;}
        public int Address {get; set;}
}
```

Action Method

```
public ActionResult Create([Bind(Include=“Name, Email, Address”)] User user)
{
        ....
}
```

Entity Framework

- Entity Framework is a database technology, which is built based on ADO.NET, that supports ORM (Object Relational Mapping) pattern
- It brides between objects and databases using Model classes



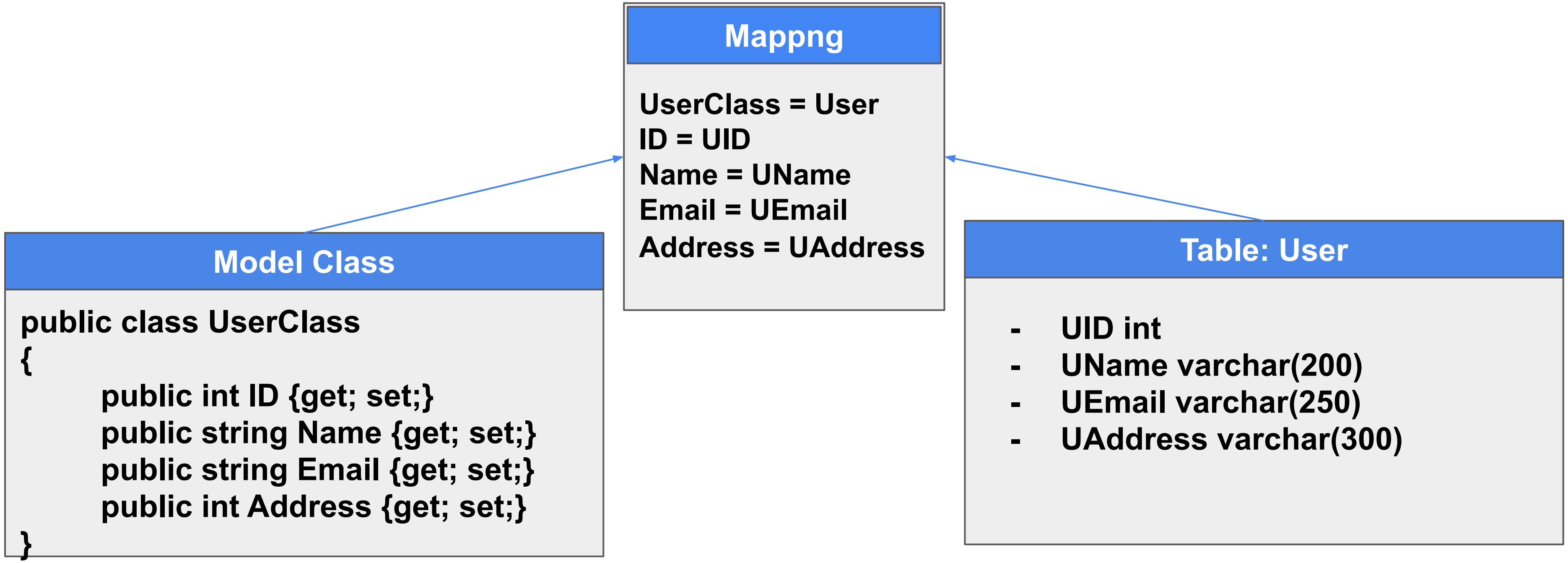
Model Class

```
public class User
{
        public int ID {get; set;}
        public string Name {get; set;}
        public string Email {get; set;}
        public int Address {get; set;}
}
```

Table: User

- ID int
- Name varchar(200)
- Email varchar(250)
- Address varchar(300)

Entity Framework



Mappng

UserClass = User
ID = UID
Name = UName
Email = UEmail
Address = UAddress

Model Class

```
public class UserClass
{
        public int ID {get; set;}
        public string Name {get; set;}
        public string Email {get; set;}
        public int Address {get; set;}
}
```

Table: User

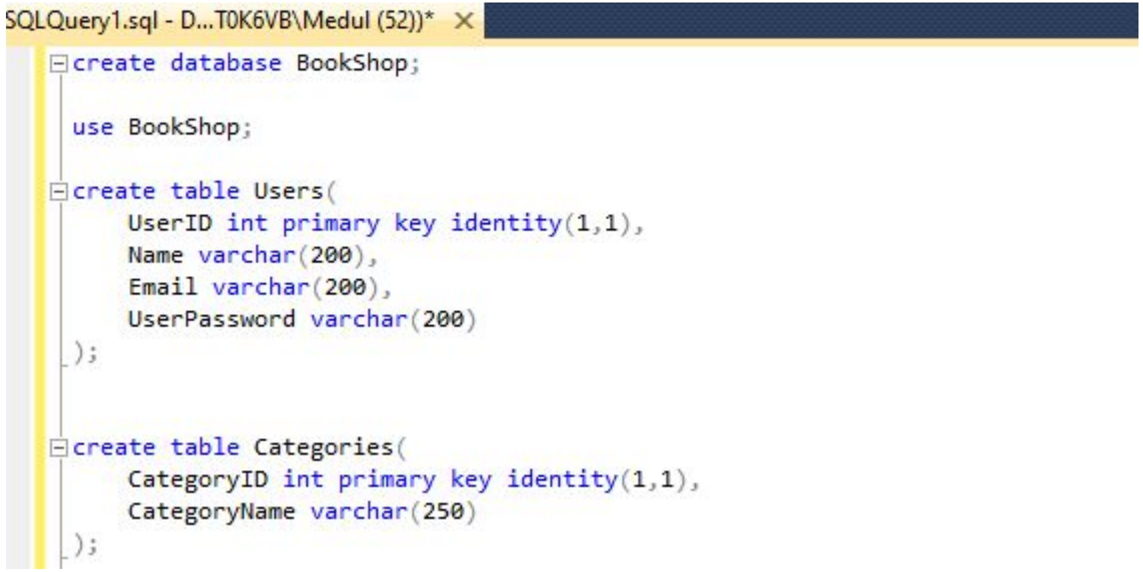
- UID int
- UName varchar(200)
- UEmail varchar(250)
- UAddress varchar(300)

Entity Framework: DbContext and DbSet

- DbContext is a class, based on which you can write LINQ queries to perform CRUD operations on table
- DbContext is a collection of DbSets
- DbSet Object Represents a table
- Generally one dataset requires one DbContext and one DbSet requires one DbSet
- A connection string is need to be created in Web.Config file to connect the database

Entity Framework: DB First Approach

- Developer has to create database first
- Model classes will be generated automatically from the Database tables
- Step - 1: Create a database first

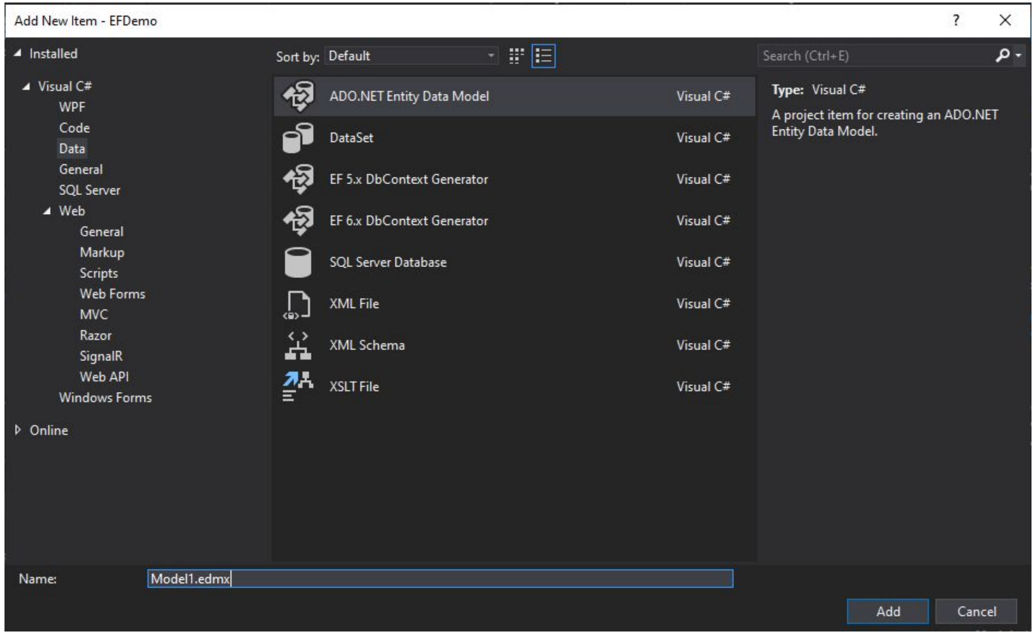


SQLQuery1.sql - D...T0K6VB\Medul (52))*

```sql
create database BookShop;

use BookShop;

create table Users(
	UserID int primary key identity(1,1),
	Name varchar(200),
	Email varchar(200),
	UserPassword varchar(200)
);


create table Categories(
	CategoryID int primary key identity(1,1),
	CategoryName varchar(250)
);
```

Entity Framework: DB First Approach

- Step - 2: Create a ADO.NET Entity Data Model class



ADO.NET Entity Data Model	Visual C#
DataSet	Visual C#
EF 5.x DbContext Generator	Visual C#
EF 6.x DbContext Generator	Visual C#
SQL Server Database	Visual C#
XML File	Visual C#
XML Schema	Visual C#
XSLT File	Visual C#

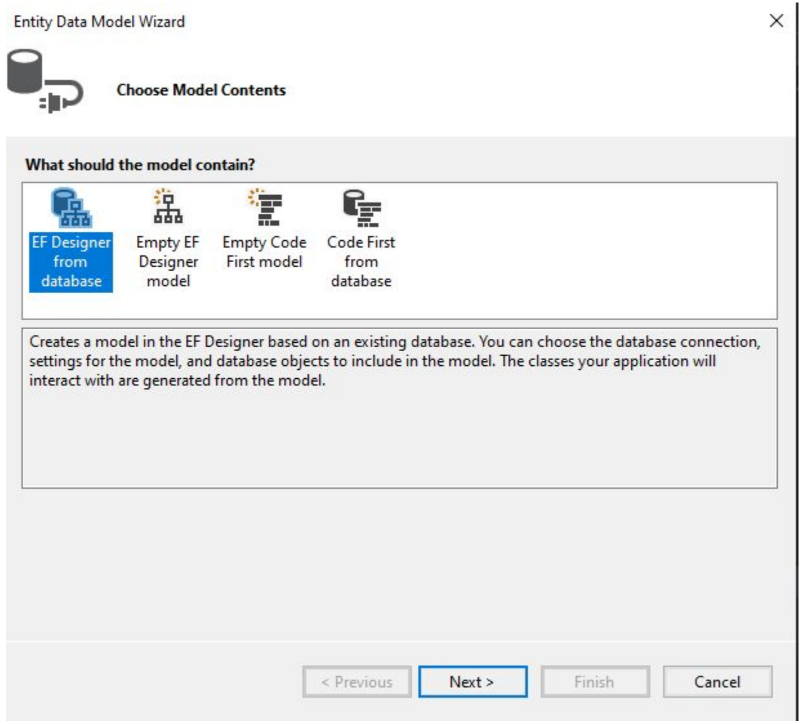
Type: Visual C#

A project item for creating an ADO.NET Entity Data Model.

Name: Model1.edmx

Entity Framework: DB First Approach

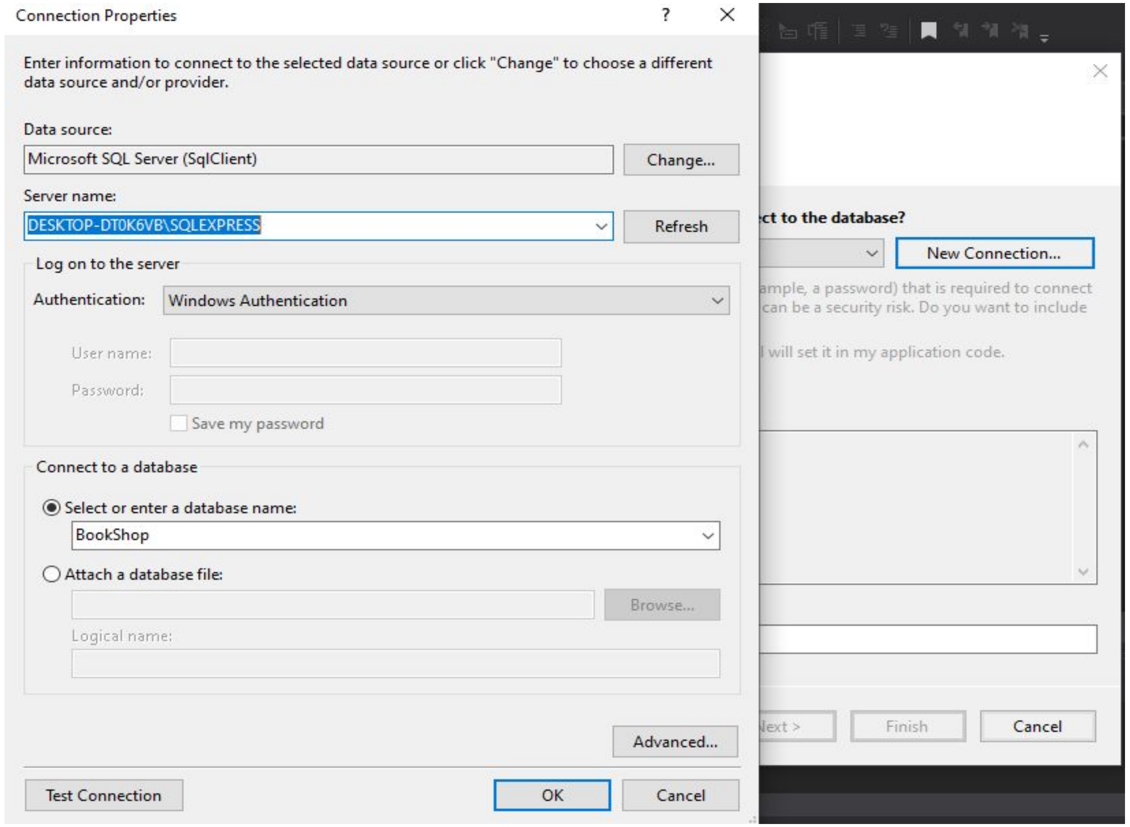
- Step - 3 : Select EF Designer from DB



Creates a model in the EF Designer based on an existing database. You can choose the database connection, settings for the model, and database objects to include in the model. The classes your application will interact with are generated from the model.

Entity Framework: DB First Approach

- Step - 4 : Set a New Connection



Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient)

Server name: DESKTOP-DT0K6VB\SQLEXPRESS

Log on to the server

Authentication: Windows Authentication

User name:

Password:

Save my password

Connect to a database

(●) Select or enter a database name: BookShop

( ) Attach a database file:

Logical name:

?
Change...
Refresh
Browse...
Advanced...
Test Connection
OK
Cancel
ct to the database?
New Connection...
ample, a password) that is required to connect
can be a security risk. Do you want to include
I will set it in my application code.
ext >
Finish
Cancel

Entity Framework: DB First Approach

- Step 5: Select entity framework version
- Step 6: Select tables for which you want to create model classes
- Done!

Code-First Approach

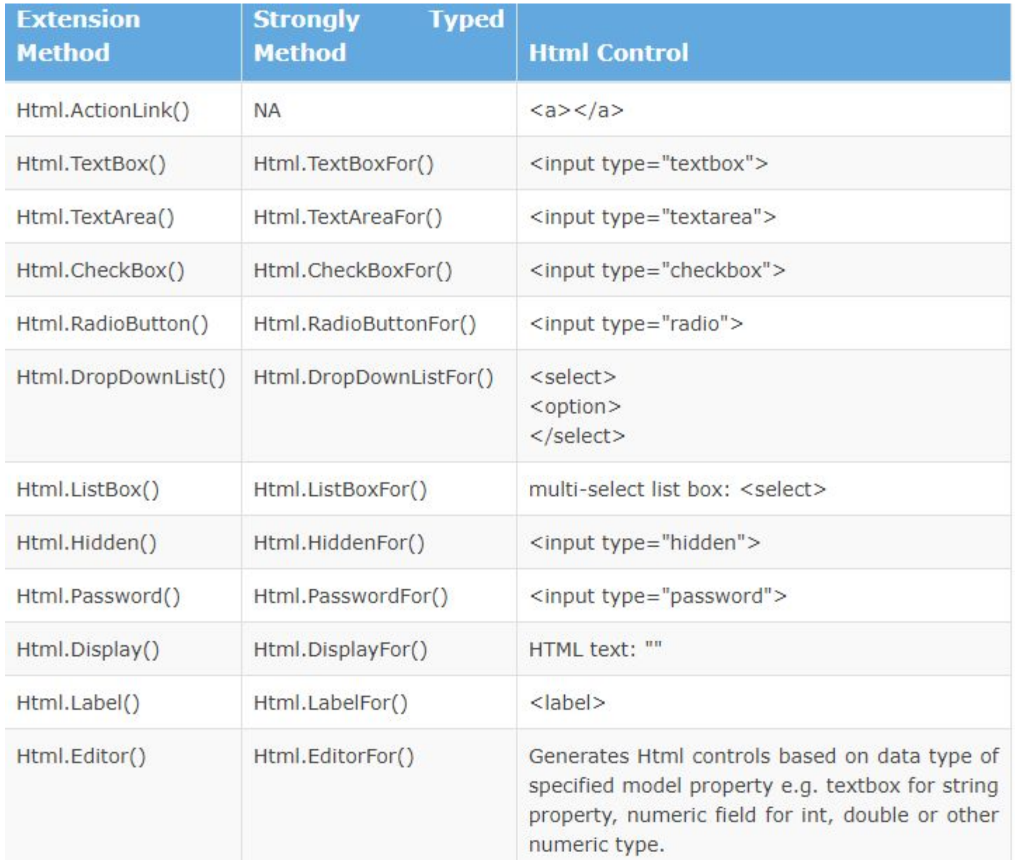
- Models are required to be created first
- Full control over the code (model classes)
- General expectation is you don’t bother with DB
- Manual changes to database will be most probably lost because your code defines the database
- Steps:
  - Create the models
  - Create the DBContext Class
  - Build your application

HTML Helpers

- HTML Helpers generate HTML elements using model class object in razor view
- Binds HTML elements to Model properties
- Assigns the value of HTML elements while submitting the form
- @Html is an object of HtmlHelper class

More at: https://www.tutorialsteacher.com/mvc/html-helpers

List of HTML Helpers



Extension Method	Strongly Typed Method	Html Control
Html.ActionLink()	NA	<a></a>
Html.TextBox()	Html.TextBoxFor()	<input type="textbox">
Html.TextArea()	Html.TextAreaFor()	<input type="textarea">
Html.CheckBox()	Html.CheckBoxFor()	<input type="checkbox">
Html.RadioButton()	Html.RadioButtonFor()	<input type="radio">
Html.DropDownList()	Html.DropDownListFor()	<select> <option> </select>
Html.ListBox()	Html.ListBoxFor()	multi-select list box: <select>
Html.Hidden()	Html.HiddenFor()	<input type="hidden">
Html.Password()	Html.PasswordFor()	<input type="password">
Html.Display()	Html.DisplayFor()	HTML text: ""
Html.Label()	Html.LabelFor()	<label>
Html.Editor()	Html.EditorFor()	Generates Html controls based on data type of specified model property e.g. textbox for string property, numeric field for int, double or other numeric type.

Validations

- Set of rules for different attributes
- Data annotations is used for validations
-

Validations

[Required]	Filed is mandatory
[MaxLength]	Min. no of characters
[MinLength]	Max. no of characters
[Range]	Value should be within min and max
[Compare]	Two fields must be same
[RegularExpression]	Pattern of value
[EmailAddress]	Email address only accepted

Validations

Server Side Validation:

- **ModelState.IsValid** -> Checks whether the model object satisfied the validation rules

HtML Helpers for Client Side Validation

- **ValidationMessageFor** -> Displays error message
- **ValidationSummary** -> Displays validation summary