ASP dotnet core from Microsoft is one of  the most popular web frameworks experienced developer Bergin Patel will teach you how to use  dotnet six, which is the latest version. Hi guys,

0:11

and welcome to dotnet mastery. My name is Bergin.  And in this course, we will walk through the basic

0:17

foundations of dotnet core application, we will  be using dotnet six in this course, along with

0:24

Visual Studio 2022 dotnet core has been a buzzword  in the industry for a while. And recently it has

0:32

been gaining more and more attention because of  the efficiency and the advantages that it brings.

0:38

So first, let's take a look at what are all the  exciting topics that we will cover in this course.

0:45

Foundation is a basic element of any  building. And that will be the first   thing that we will learn as we explored the  fundamentals of dotnet core and its evolution,

0:57

then it's critical to understand what is new in  dotnet six when it comes to an MVC application.

1:04

So I will highlight the new features in dotnet  six with respect to the MVC application,

1:11

we will be using Entity Framework  core to setup database connection   string and perform all the  database related operation.

1:20

Once we have Entity Framework core configured, we  will be performing CRUD operations on any model.

1:27

Crud stands for create, read, update and delete.  So while we perform the CRUD operations, we will

1:35

also understand how client side and server side  validations work with dotnet core. We will display

1:42

nice alerts using temp data and toaster J S. Now  while we learn all of this one critical piece is

1:50

error solving when it comes to any programming,  I will display you some of the common errors,

1:56

and how will you approach those errors and resolve  them. Lastly, we will deploy our application to

2:03

Azure. And we will see everything live including  SQL Server, which will be hosted on Azure as well.

2:11

So you can see there is quite a few things that we  will be learning in this course. Let's get started

2:17

with all of that from the next video. In this  video, let me show you the demo application that

Live Preview

2:25

we will be building. We will start with an empty  project. And then we will use boots watch theme

2:32

to add a nice styling. Next we will be building  a category crud project. So when you click on

2:39

category in the navigation bar, it will open up  the page and it will display all the categories.

2:46

First thing that you notice here is we have a  create Category button. When we click there,

2:52

it takes us to category create. And  here without populating anything,

2:57

if you press the Create button, we have default  client side validation. Not just that we also

3:04

have some server side validation. Like if we enter  both name and display order to be same and hit the

3:11

create button, we have a server side validation,  that display order cannot exactly match the name.

3:19

Then we have a backdoor List button here to go  back to our index page with the category list.

3:26

We can edit any of the existing category. And  as soon as you edit or create something, we have

3:32

nice toaster alerts that you can see. Lastly,  we will implement the Delete functionality.

3:38

And on that also, we will have our alerts. So  we will see server side client side validation,

3:45

we will implement toaster alerts and take a look  at temp data. Not only that, once we complete

3:52

the application, we will be deploying that to  Azure. So that will give you a complete flow

3:58

of the application as we deploy things to Azure  along with our database. So with that, let's start

4:06

on this from the next video. Let's take a look  at all the tools that we will use in this course.

Tools Needed

4:14

We will be using dotnet sex for our  application. So make sure to install

4:20

the latest version after add sex. For  the IDE we will be using Visual Studio.

4:27

The version that I will be using is Visual Studio  2022. There are lots of advanced features in 2022

4:35

as compared to 2019. So you can always install  the free version of Visual Studio 2022.

4:43

For our database, we will be using SQL Server.  So first you will have to install SQL Server

4:50

and then you should install SSMS which has  SQL Server Management Studio. If you have

4:56

older version of SSMS that's completely okay.  As long as you're able to connect to SQL Server

5:03

on your local machine, we will be using that  connection string, and everything will work.

5:09

So once you've installed all three of these  software's, let's continue from the next video.

Introduction to .Net Core

5:17

This video, I want to walk you through the journey  of dotnet core dotnet core is probably the biggest

5:24

change that the dotnet language has encountered.  In 2002, Microsoft introduced web forms,

5:32

which was a revolution at that time, web forms has  its drawbacks, and there was a need to overcome

5:39

all of them. Because of that, the dotnet team came  up with a new architecture, which was dotnet MVC.

5:48

Now, even though I love MVC, and I have built  many applications in MVC, it had its flaws,

5:56

like it was created on top of the components for  web forms. Because of that, it was tied to IIS,

6:04

and ultimately, Windows operating system. But with  the evolution of web development, Microsoft had to

6:12

keep up with the changing technology. Finally, in  June of 2016, Microsoft released ASP dotnet core,

6:20

and it was the first version dotnet core is built  on top of the new dotnet core framework, it is

6:27

completely rewritten, and it is a cross platform  version. Hence it is not tied with Windows.

6:35

Also dotnet core was built with cloud in  mind. So it is extremely robust with that.

6:42

Then in August of 2018, Microsoft released  dotnet core two, and the team has been

6:48

active with releasing new versions, there  was a big change from dotnet 2.1 to 2.2,

6:56

because we had to update quite a few class  libraries. And there were a few challenges. But

7:02

since then, dotnet core team has been releasing  new versions with three 3.1 and dotnet five, which

7:10

was released in November of 2020. After that, that  is start at six which will be released in November

7:17

of 2021. We will be using the Preview version, but  whatever we learned will be the same once dotnet

7:25

six is released. So this is a small overview of  all the dotnet frameworks and their evolution.

7:33

That being said, why should when you start NET  Core as compared to the classic dotnet dotnet core

7:40

comes with many advantages. First one is  ASP dotnet core is fast and open source.

7:47

If you compare that to the traditional  dotnet applications, that have been quite   a few benchmarks, and it is very fast, when you  compare that to web forms, or even dotnet MVC

8:00

dotnet core is also cross platform, the classic  dotnet was tied to IIS and Windows. But since

8:08

the dotnet core is rewritten, it has removed that  dependency with dotnet core. We also have a built

8:15

in support for dependency injection, which saves a  lot of time and it is extremely helpful. Once you

8:23

get used with using dependency injection, you  cannot imagine your application without that.

8:29

With any programming language, it is critical  that the new updates or the new version that

8:34

are released, they should be easily upgradable.  And that is one of the feature with dotnet core.

8:41

When a new version is released, updating to that  new version does not have groundbreaking changes.

8:48

Because of that you can always keep up with the  new versions. dotnet core is also cloud friendly.

8:56

When the dotnet code was being written,  cloud architecture was kept in mind.

9:01

And because of that it is completely compatible  with all of the cloud components. And lastly,

9:08

when it comes to performance dotnet code  exceeds all of the previous versions, and even

9:13

the new versions in dotnet code that have been  released. They supersede the previous version.

9:20

The code actually gets more optimized that results  into improved performance. The ASP dotnet core

9:27

compiler will eventually optimize the entire code.  Whenever the code is re composed using dotnet

9:35

core framework dotnet course actual performance is  multiple times than any of the frameworks previous

9:43

implementation. Because of that, it is clear  that Microsoft has a long term plan with dotnet

9:50

code technology. So with that brief overview,  let's continue our learning in the next video.

Dependency injection

10:00

In this video, before we take a look at the other  files, I will introduce you to a new concept,

10:06

which is dependency injection. ASP dotnet core  implements a simple built in dependency injection.

10:14

Container Dependency injection is an integral  part of the ASP dotnet core architecture. dotnet

10:22

core injects objects of dependency classes through  constructor by using the built in IOC container.

10:30

Before I show you what advantages dependency  injection brings, let's see a scenario where

10:37

there is no dependency injection. In a typical  application, let's see we have three pages right

10:43

now. And we have some common functionality that we  want to use across all the three pages. Like let's

10:51

say we want to send emails, and we want to access  our database in all the three pages that we have.

10:59

Now, let's imagine that on these three pages,  we need to access the database first. So what we

11:05

will do is we will create the object for database  classes on all the three pages, we will have to

11:11

open that connection, we will have to do the  database operation. And then we will have to

11:16

close the connection in all the places to do the  same and create object for email implementation

11:24

that we have in all the three pages. So you can  see this is a lot of duplicate code. On top of

11:31

that, what happens in future, if you change the  implementation of how you access the database,

11:37

or email, based on the current configuration, you  might have to make that change in all of the three

11:43

pages, which is a big mess, because right now,  it's three pages down the line, it could be 30,

11:50

or 300 pages. And the issue here is that on each  page, you will deal with creating the object

11:58

managing them as well as disposing them. And  that will be a time consuming effort when we

12:05

have to do that in all the pages. So that being  said, what is a solution to avoid all of this

12:12

and get an optimal architecture. The answer  to all of that is dependency injection.

12:19

First, let me show how this scenario would look  like. Again, we will have the three pages. And we

12:25

will have email and database functionality. These  are the common things that we had before as well.

12:32

But now we will have something special, which  is dependency injection container. So as you

12:39

can see, we have got our dependency injection  container that will have an ID email and an IDB

12:46

interface and its implementation. So inside our  container, we have the implementation of the eye

12:53

email interface and the IDB interface. When any  page will need access to these functionalities, it

13:01

will just ask the dependency injection container  to create an object of this functionality

13:08

and directly give page an object to use. So inside  the page, we will actually be using an interface.

13:16

And then dependency injection does audits magic  of passing the object when the website needs it.

13:23

That way, we do not have to deal with creating the  object disposing or managing that object inside

13:30

our pages or pages will look very clean with just  the interface. All the instance and implementation

13:38

will be done by dependency injection container.  Now in future if you want to change or replace the

13:45

email class, you do not have to make any changes  in the pages, all you have to do is just change

13:52

the implementation inside the email class. And  since we are registering that in the container,

13:59

next time when we build the project, it will  take the new implementation. So you can see we

14:05

only have to change in one place. Now. That is one  of the main advantage that comes with dependency

14:12

injection. Now in order to use dependency  injection, you can use many third party tools.

14:18

But with dotnet core, we have a built in  dependency injection container, and that has its

14:24

own advantages. So I hope that you have a short  overview of how Dependency injection is helpful.

Create Project

14:34

First thing that we have to do  is we need to create our project.   I will be using Visual Studio 2022. Here we have  the recent projects on the left and on the right

14:46

hand side. We will catch start there. We want to  start by creating a new project. So we will click

14:53

that and it will display all of the templates. You  can either navigate and try to find the templates

15:00

or you can just search here for model, views  and controller. As soon as you type that,

15:07

you see there are two templates that are available  right here, we will be using the ASP dotnet core

15:14

with C sharp language, we do not want the  other language. So select the C sharp language,

15:20

or ESP dotnet core web app. And we have model  views and controller. Let's hit the next button.

15:28

And we have few more options that we have to  configure. Before writing the project name, we

15:35

can give our solution a meaningful name. So let's  say we are making our project for bulky book.

15:42

And in that you can have multiple projects  like that can be data access layer, that can be

15:47

business layer that can be web layer, and so on.  So solution name, we can keep that as bulk ebook,

15:55

and then the project name. Let's call this bulky  book Web. That way you can keep the project name

16:02

and the solution name separated. Let me change the  location to where I want to save on my computer,

16:10

and we will hit Next button. Final thing that we  have to select here is the Framework version. We

16:17

will be using dotnet six, so we will select that  in the first option. Next we have authentication

16:24

time. When you clip the authentication drop  down that are individually users account,

16:29

Microsoft Identity platform and Windows.  Right now we want to keep things simple and

16:36

understand the basic folder structures of when  an empty dotnet core MVC project is created.

16:43

So we will select None inside the authentication,  we will keep the Configure for HTTPS selected, we

16:50

did not want to enable Docker right now, because  we do not want to work with the containers.

16:58

With this configuration, let's create our project.  It will take a while right it will create the

17:05

project and it will load all of the files and  folders. Great. So once the project is created,

17:13

it will load the complete view of Visual Studio.  On the left side here we have the main panel where

17:20

we will be coding everything. And on the right  side, we have Solution Explorer, good changes

17:26

if we want to commit and some properties. Now by  mistake if you close the Solution Explorer here,

17:33

do not panic, you can open it back up by going  to views here. And we have the first option,

17:40

which is Solution Explorer. With that in  place. Let's continue from the next video.

Project File

17:48

Now that our project has been created, let's  take a look at the files and folders that we have

17:54

by default. The first thing that we should  take a look at is the project file. In order

18:00

to access the project file, you will right  click on the project name or the solution   we have our project with the website  icon here. If you right click there,

18:10

we have the added project file. When you open that  up, you will see few configuration right here.

18:18

The first section here defines what is  the target framework that we are using,

18:24

we are using dotnet six, so you can see that right  here. The next important thing inside this file

18:31

is the item group item group will contain all the  new get packages that we are using in the project.

18:39

We selected the runtime compilation, and that's  why that package has been installed. In future

18:46

we will be installing more packages when we  connect to database use Entity framework code

18:52

and so on. So in that case, when we add new get  packages to our project, a new entry will be made

18:59

inside the CS proj file or the project file. Now  usually you do not work with the project file.

19:07

But it is always good to know that we have all the  packages and the references listed in the project

19:13

file. If in case you need to access that. So  that covers our first file, which is the CS proj

19:21

or the project file. Now that we have seen the  project file, the next thing that I want to show

Launch settings

19:28

you is we have dependencies here and we can see  the packages that we saw previously. Inside the

19:34

project file will be installed right here. We will  be adding more packages down the road. But the

19:42

next folder that we have is the properties folder.  And inside there we have launch settings dot JSON.

19:50

In here we have the different profiles using  which we can run our application. Like you

19:56

can see inside the profiles, we have a bulky book  Add profile, and we have an IIS Express profile.

20:04

If we use the IIS Express profile, we know the  port number that will be used here is explicitly

20:11

defined. If we use the bulky book web, in that  case, it will use 5005 1001. Based on HTTPS or

20:20

HTTP, the bankbook web is the default time, what  this will do is it will run a dotnet command line,

20:28

and that will trigger the application. So if you  try to run this directly with by the key book web,

20:35

you will first see a command prompt. And  then it will launch the website on port 5001.

20:41

You can see inside the logs it is displaying all  of that. So that is one way or one profile to

20:48

run our application. If you hover on this down  arrow, you will see there is another profile,

20:55

which is ies Express. And that is the  name that we have in launch settings.

21:01

If you run with IAS Express, it will use the port  number right here for the SSL. So let me run that

21:10

and show that as well. Great, you can see it  runs on the different port that is defined

21:17

right here. So the default behavior is using the  bulky book web profile. But we will be changing

21:24

that and using IIS later on. So that was a  brief overview on non setting starting JSON.

21:33

After that the next folder that you see here  is the www root folder. If you expand that,

21:40

you will see all the static files of your project.  So any static files like CSS, JavaScript, images,

21:49

or any libraries, everything will go inside the  www root folder. The www root folder will not have

21:58

any C sharp files. This folder is only meant  to serve the static files of our application.

22:06

So we will be using the static folder extensively  throughout the course, when we are adding some

22:12

GS or some images, or any other libraries. The  www root folder will be the root folder of your

22:21

application. So always remember, if you ever  have to add any static file, it will always

22:28

go inside the www root folder. Then we have  the folders for controllers, models and view,

22:36

I will get back to them in a couple of videos.  But the file after that is app settings dot JSON.

22:44

This is the file in which we will be adding  all of the connection strings and secrets

22:50

of our application. Like you might have some  API keys, you might have some SendGrid keys,

22:56

you have stripe payment keys, any of the  static secret keys that you want to save,

23:02

we will be storing them inside app settings  dot JSON. If you expand this app settings,

23:07

you will notice that we have App Settings dot  development dot JSON. So you can create new

23:14

JSON files and it will automatically bundle  them inside app settings dot JSON. Like if

23:20

you create for another environment,  App Settings dot staging dot JSON,   then you can have App Settings dot production dot  JSON, all of them will be bundled in one umbrella.

23:32

And then based on the environment variable,  you can configure it to use the different app

23:37

settings file. Because connection string for a  database in development will be different if you

23:43

compare that to staging, preview or production.  So that way, you can go into those configuration,

23:51

we will be using just app settings dot JSON  right now, because we will be working with

23:56

the local host. Also in production, there are  multiple ways of saving secrets like you can

24:02

add them to Azure Storage world and much more.  But that is beyond the scope of this course.

24:08

To get started, you need to remember,  all of your application secrets   must be inside settings dot JSON, and not  directly inside any of your CS our class

24:20

files. We will come back to this file in the later  videos and we will add our connection string.

24:28

Now the last file that we want to take a look at  s program.cs. Let's do that in the next video.

Programs

24:37

Next we have program.cs this is the file that will  be responsible for running the application. Once

24:44

we open program.cs You can see we have a variable  builder where the web application dot create

24:51

builder is passed with the built in arguments.  When you run with the dotnet command, you can

24:58

pass custom arguments If you want with that, it  will configure the application, and it will create

25:05

the Web Application Builder object. Now in the  previous video, we saw that we can use dependency

25:13

injection with dotnet core. When we want to  register anything with our dependency injection

25:19

container, we will be doing that right here. So  let's say if we want to register our database or

25:26

email or anything else, we will have to do that  between the builder. And before we call build

25:33

on that builder object. So right here, we are  just adding one service to the container, which

25:39

is builder dot services dot add controllers with  views, we are adding the service to the container,

25:46

because we are using MVC application for our  project. If you were using razor pages, then

25:53

the service will be different. Now in the future  videos, when we configure database in our project,

26:01

and we add that to dependency injection, we will  be adding a new service here in our container, our

26:07

DB context. If you are working with any version,  prior to dotnet, six, or even some of the initial

26:15

preview versions of dotnet, six, then this file  was divided into a separate startup.cs class file.

26:24

And the services that we add to Container were  inside a method configure services. And everything

26:30

from line nine onwards was inside a configure  method. So what we have on the top is we will be

26:38

adding services to our container, then we need to  configure request pipeline. And that pipeline will

26:45

be configured from the highlighted section of your  mind might be wondering what is this pipeline,

26:52

the pipeline specifies how application should  respond to a web request. When your application

27:00

receives a request from the browser, that request  goes back and forth through the pipeline. Let me

27:06

switch to the presentation here. We have different  browsers here. And then we have a pipeline.

27:13

The pipeline specifies how application should  respond to a request that is received. When

27:20

your application receives a request from the  browser, that request goes through the pipeline.

27:26

In the pipeline, you can add items that you want.  Pipeline is made up of different middlewares.

27:33

And MVC is a type of middleware itself. So if  we want an application to be built using MVC,

27:41

we have to add that middleware. Other  example could be authentication, middleware,

27:47

authorization, middleware, and so on. What exactly  happens is when your request will go through each

27:55

of the middleware, it gets modified by them, and  eventually it is passed to the next middleware.

28:02

If that is the last middleware in the pipeline,  the response is returned back to the server.

28:08

Let's take a look at the few middlewares that we  have in our application. Let me switch back to the

28:14

code here. And you can see in the pipeline, first  we are checking if it is development or not in the

28:21

environment. If it is, then we are adding the  use developer exception page that will show you

28:28

user friendly exceptions, so that you can debug  and solve them. But if it is not development,

28:34

then we are just redirecting them to an error  page. The next middleware is HTTPS redirection.

28:43

And then we have a middleware to use our static  files that are defined in www root folder.

28:51

We also have a routing middleware, and we  have authorization middleware, and we add

28:57

authentication to our project, we will have to add  a new middleware inside the program.cs as well,

29:05

then we have using a map controller route that  will map the different pattern that we have

29:10

for MVC. Based on this routing, it will  be able to redirect a request to the

29:17

corresponding controllers and action, then  you should always keep in mind that order of

29:23

pipeline is extremely important. The way  you write middlewares in the pipeline,

29:29

that is exactly how the request will be passed.  So first, routing will be done. And it checks for

29:35

authorization and so on. So in this scenario, if  you want to use authentication to your pipeline,

29:42

we have a middleware, which is app dot use  authentication. But if you do this, then it won't

29:49

work. Because authentication middleware should  always come before you authorize a user. Because

29:56

you only authorize the user that is authenticated.  That is the basic fundamentals of authentication

30:03

and authorization. So if you place the pipeline  in some different order that will break things.

30:10

Inside the endpoints. Here, you can see we have  a controller name and action name, and some

30:16

ID. This controller route will make more sense  when we understand routing. In the next video.

MVC Architecture

30:25

Let me show you more details on how routing  works. I won't be running the application to

30:31

show you that, but I want to walk you through some  theory. You can see when it comes to routing in

30:38

MVC application, we have controllers, and we  have actions. Before we explore the routing,

30:45

let me walk you through the main components  of an MVC application, which is models,

30:51

views and controller. Let me switch back to  the presentation and give you a brief overview.

30:59

Now, if you remember, we had three folders for  models, views and controllers, and that is what

31:05

MVC stands for. The first thing in MVC is model,  which represents the shape of the data. A class in

31:15

C Sharp is used to describe a model. The model  component corresponds to all the data related

31:22

logic that user works with. Let's say inside your  application, you have a table that stores all the

31:29

category or all the product details than that  product will be a model itself. model basically

31:37

represent all the data in your application, it can  be a table that you are storing inside SQL Server,

31:44

or it can be a model, which will be a combination  of multiple tables, and so on. This model can

31:52

either represent the data that is being  transferred between views and controllers,

31:57

or any business related data model that will  represent all the tables of the database. So

32:04

if you have 10 tables in your database, we will  have at least 10 models that corresponds to them.

32:12

There is also more complexity, but we will go into  those details later on. Right now you can think of

32:19

all the tables in your database will be a  class file, which will be a model than all

32:25

the properties of that class file will be the  columns of the table. That is a simple relation

32:32

that you can think of right now. Then we  have view in an MVC, which is the user

32:39

interface. You can be tired of HTML and CSS that  you write, to make things fancy and beautiful.

32:48

Whatever you see on the website with your eyes, is  basically the view that is being displayed to you.

32:55

But now you need to think of what happens if in  a website, you have a button and you click that

33:00

button. What happens is that view will interact  with your model to display some of that data. But

33:08

view does not interact directly with the models.  For that we have something known as controller

33:15

controller acts as an interface between model  and view to process all the business logic and

33:22

incoming request. So controller acts as an  interface between model and view to process

33:29

all the business logic and it manipulates that  data using model and interacts with the view

33:36

to render the final output. This is just a brief  overview of how model views and controller works.

33:45

So let's say if a user clicks on a button  controller is the first thing that will receive

33:50

that request and controller will have lots of  action methods based on those action methods

33:58

controller will redirect the request to one of the  action method and controller will use the model it

34:05

will fetch all the data that it needs to display  inside the view. Once the view is rendered,

34:11

it will pass all of that to the controller and  controller will then pass a response which will be

34:17

sent back and the user will finally be able to see  the page. So you can see controller can be treated

34:25

as heart of your application. That is where  we will have all the logic of your application

34:32

and it is the one which will be interacting  with models and views. So with that in place,

34:38

now you see that the request first comes to the  controller and it's action methods. So with that

34:45

general idea if we go back you can see inside the  map controller route, we have a pattern where we

34:52

define a controller and an action method. So  here we are saying that the default if nothing

34:58

is provided It should go to the home controller,  and it should call the index action method.

35:06

But that is a 10,000 Fate overview. Let's take a  look at routing a little more with some theory.

Routing Overview

35:14

Before we see routing in action, let's see  routing with some examples. You can see here

35:21

we have a general pattern of routing. The first  thing highlighted in blue here is the domain of

35:27

the URL. When we run on the local computer,  you will have a local host and a port number.

35:34

Whatever it is, after that port number will be  the route that we want to use when we are calling

35:41

a page to be loaded. In the first example,  you see we have something called as category,

35:47

then we have an index and some number. When we  are working with MVC, after our port number,

35:54

or domain, whatever is the first thing that  we have will be the name of the controller,

36:00

then the next forward slash after that will be  the action of the controller. And after that,

36:07

if you have something that will be the ID.  This is the pattern of routing with MVC.

36:14

If we go back to the application, you can see  the same format right here. First we have the

36:20

controller name, then forward slash, we have an  action name, then forward slash, we have the ID

36:27

that corresponds with the request that we have  here. Keep in mind that ID is an optional field,

36:35

controller and action are not optional. But if  they are not defined, we have set a default route,

36:41

that if there is no controller and action, you  can use home controller and index action as the

36:48

default route. Because of that, we have our home  controller. And we also have the index action,

36:55

which I will show you. Before we dive into those  details. Based on the understanding that we have

37:01

here, I have given some sample URL, I want you  to try to find out what will be the controller,

37:09

action and Id based on this URL, I can remember  if controller is not defined default, one that

37:17

we have in our application is let's go back the  home controller. And if the action is not defined,

37:23

that will be index action. So based  on that the first URL that we have,

37:29

the controller name is category, the action is  index. And we do not have any ID. For the next

37:37

one, we have controller name as category, we do  not have any action. So index will be the default

37:45

action. And finally, ID is no. Next what we have.  For the third one, we have controller as category

37:54

action as added and ID as three. The last one,  we have controller product action details,

38:02

and we have ID as three. So with that if you get  a URL, now you can identify what is the controller

38:10

name, what is the action name, and if there is  an ID or not. So with that brief understanding of

38:17

routing, let's actually run our application,  see routing in action in the next video.

38:26

Now we want to understand routing in MVC. This  is one of the tricky topics that I had when I

38:33

first started learning MVC. So I want to make  sure that you get familiar with the routing.

38:40

Before we see the complete routing in  action. Let me walk you through the   three folders that we have. We have a folder  for controller, we have a folder for models,

38:50

and we have a folder for views. By default, we  have a home controller that has been created.

38:57

models will be all the data related  models that people want in the project.

39:03

So let's say if you are dealing with products  that you want to display on the page,   you will have a product model and we will be using  that model in controller and views. Right now you

39:14

can just think of models as tables that you want  in your database. If you want a product table,

39:21

you will have a product model. That is not always  the case, but we will explore models later on.

39:29

The main thing that we want to work on  is understanding controllers and views.

39:35

Controller is the heart of the application. At  the same time views is what will be displayed on

39:41

the screen when user is looking at the page. Now  the way navigation works is when we have a home

39:48

controller, all of the views or UI pages that are  displayed with the home controller will be placed

39:55

inside the same folder name as the controller  name So you can see they have also created a home

40:02

folder by default, that is also a convention with  the naming of controllers, it should always end

40:09

with the keyword of controller. That is how the  application will know that this is a controller.

40:16

If we expand our home controller, you will see  some code right here, we have a class with the

40:23

name of home controller, and it implements  the default or the base class of controller.

40:30

And right here we are registering the logger using  dependency injection. Do not go into that detail

40:37

right now, I will explain dependency injection in  much detail. But for now, to understand routing,

40:45

you can see we have two action methods. Inside  controllers, you can have multiple action method.

40:53

If we go back to the URL here, you can notice  that we have the controller name and action name.

41:00

If the URL was forward slash home, forward slash  privacy, then it will go to Home controller look

41:07

for a privacy action method. And it will load the  content that we have here. The return type for an

41:15

action method is I action result, because I action  result is an abstraction for multiple return type,

41:22

it can return a view, it can redirect to some  action method, or it can redirect a page. And much

41:29

more than I said that if the URL is home forward  slash index, it will return the view that we

41:36

have defined here. Now you might be thinking that  where is this view we are talking about that view

41:43

will be inside the views folder. The way it finds  our maps, the view for this index action method

41:50

is inside the views folder, it will look for the  name of the controller, which is home controller.

41:56

So we have that folder. Inside there, we will have  a view with the name of this action method. And

42:04

it is index action method. So that will be mapped  to this particular view. If we open that view,

42:12

we have some HTML and Bootstrap classes. So  here, nothing fancy is going on, we are just

42:19

displaying some text. So let's run our application  and see routing in action. Right now it will run

42:28

in the command prompt window. And once that  is done, it will run the website on port 5001.

42:36

In the URL right now, nothing is present. What  should happen if there are no controllers and

42:42

actions in the URL, we define that inside the  program.cs. We said if nothing is present,

42:50

default that to home controller and index  action method. So what we see on the screen

42:56

is from the home controller index action method  to confirm that if you go to the home controller,

43:04

you can add a debugging point by clicking anywhere  on the window. What that will do is when it hits

43:11

this controller, this will be yellow. So if we go  back, and if we refresh the page, you can see it

43:18

hits our breakpoint here. So that means if nothing  is present in the URL, it is calling the home

43:24

controller index action method. Let me also add a  debugging point inside the privacy action method.

43:33

So we will hit continue here and the request goes  back. So we will hit continue. And then here we

43:41

have returned view that will find out the view  that is associated, and it will display that on

43:47

the page. If you right click on the view here, you  have linked to go to the view, if you click that,

43:54

it will automatically redirect you to the index  view because it knows it has to find the home

44:00

folder inside there, there will be an index view.  This is how controllers and views are associated.

44:09

So if we go back to the application, while this is  running, we have the Privacy tab here. And you can

44:15

see on the bottom right the URL that it will go to  is home forward slash privacy. So what will be the

44:22

controller name, controller name will be home and  action name will be privacy. So it will go to the

44:30

home controller. It will go to the privacy action  method and it will return the view. The view is

44:37

inside home we have privacy and it should display  this particular paragraph. Let's go back and hit

44:45

that. It hits our debugging point. That is great.  Let's continue and it displays the privacy policy.

44:54

So that is perfect. With this you can see  how routing is working in action. With MVC

45:01

based on the URL, you have to define  the controller name and action name,

45:06

it goes to the controller points out that action  method. And it returns a view based on the view

45:13

that you have defined inside the views folder.  Now, if this is too much right now, do not worry,

45:21

just understand the basics that I have taught you  in this video. But as we proceed with the course,

45:27

routing will make much more sense. And  if this is too much for you do not worry,

45:33

you are not alone. When I was learning MVC, about  10 years ago, at that time, I had a very hard time

45:40

trying to understand this since I  was coming from the webform world,   where we just have code behind files. So as we  program this more, it will get much more familiar.

45:52

And you will love the way it works. So  let's continue from the next week here.

Routing in Action

45:59

Now that we have seen the basic controllers and  views and how the interaction works with routing,

46:05

I want to walk you through the basic views that  are available with our project. We saw the home

46:12

folder inside views that corresponds  to the views of the home controller.

46:17

But on top of that, we have something called a  shared shared folder is used for partial views.

46:24

And partial views are similar to user components  if you are coming from classic C sharp.

46:31

So it's basically a view that you can call within  a view in multiple places in your application.

46:38

Along with that, we have a special partial  view, which is underscored layout. And this

46:44

underscored layout is the default master page  of your application. So if you open that up,

46:51

you can see we have some styling on the top here,  we have a header right here. And we have a div

46:58

where we render the body, since this is the  master page, so whatever we display inside the

47:04

other views, it will use this underscored layout  as the default master page. So inside index,

47:11

when we are displaying this content, it was  displaying that with the navigation on the top, we

47:17

also have a footer here. And we have some common  JavaScript that we want across the application.

47:25

We have the main HTML and body tag. And we have  head right here, where we are adding the styling.

47:33

We will be updating this in future videos.  Whenever we want to add some CSS and JS

47:39

globally. This is the place where we will be  adding that next we have elation scripts partial,

47:46

and this is a partial view where we are just  adding scripts for some validations. Wherever

47:53

in some views, let's say in index view, we want to  use validations, then we will include this partial

47:59

view on that page. So that way, we do not have  to write those script tags, we will just include

48:05

this partial view, and that will be included.  We also have an error partial view that will be

48:13

used to display the errors. If you encounter in  the application. We will be adding more partial

48:20

views as we proceed with the course. So do not  worry. But remember shared folder will contain

48:26

all the partial views. And underscore layout  is the master page of your application.

48:33

While the application is running, if I switch  back quickly, you can see on the privacy page,

48:38

we have the header here, we have the footer here,  and we have the body. So where we have this render

48:45

party at that place, whatever we had inside the  privacy, these two lines are being displayed. So

48:53

I hope that makes sense. Then we have something  called as underscore view import and underscore

49:00

view start. Let me open the view import first.  And we have the global namespace right here.

49:08

So inside your application, let's say you  want to access some namespace in all of the

49:14

pages. If you add that USING statement here,  it will be accessible across all the pages,

49:20

controllers and classes in your project. That way  you do not have to type this namespace every time.

49:28

We will see that in action as  we proceed with the course.   But one important thing that you see here is tag  helpers. helpers are bindings that are provided

49:39

by the dotnet core team that looks like HTML  tags. But they are special tags that are adopted

49:46

by the Microsoft team from other languages. After  looking at the success like Angular and react,

49:54

I will show you one example quickly here  in the underscored layout. If we go back to

49:59

our application we'll click on privacy to see it  builds the URL home and privacy. Here we know that

50:07

home is controller and action name is privacy.  So how do you think this link is provided on this

50:14

navigation, it is provided using a special Tag  Helper. Let me expand the header here. And you

50:22

can see we have an anchor tag. And then we have  tag helpers. For ASP controller and ASP action.

50:30

Tag helpers will start with a prefix of ASP  hyphen, and then the name. So here we have

50:37

attack helper, which says ASP controller, and  then we define the controller name for routing.

50:44

And in that controller, what action name should  be calmed. So that is ASP action. Do not worry

50:51

about the ASP areas for now. We will be  using lots of tag helpers that are provided

50:58

by the dotnet team in future by tag helpers  have been included in the project. And we

51:04

define them globally. Inside the view imports  file. The last file inside the views folder

51:11

is the view start. This file will define what  is the default master page for your application.

51:19

In set privacy, you see we have not defined what  will be the master page. But it is by default,

51:25

using underscored layout, because that is what  has been defined inside the view start file.

51:32

If you want to explicitly define  a master page for privacy,   which is different, you can do that directly  by defining that on the top. And that will take

51:42

preference over what is defined in view start  but you start will have the default master page

51:49

for the application. Now I know the section  was a little bit lengthy, but this overview

51:55

was critical when it comes to understanding  the default views that have been provided.

Tag Helper

52:03

In this video, let's take a look at tag helpers.  Tag helpers are brand new to ESP dotnet core.

52:11

Microsoft looked at the success around libraries  like Angular, React, and decided that implementing

52:19

an Angular directive like experience in the new  ASP dotnet was so important to the adoption of

52:27

dotnet core that they decided to create tag  helpers from crowns up. Even though there are

52:33

similarities between Angular directives and tag  helpers, there is a major difference. Tag helpers

52:40

are for server side rendering, while the Angular  JS directives are all about client side rendering.

52:48

If you have worked with older versions of dotnet  core, we had something called as HTML helpers.

52:55

They are still around a tag helpers are being  modern with the tag like approach. So it's much

53:02

user friendly. Tag helpers are very focused around  the HTML element and are much more natural to us

53:10

as compared to HTML helpers. We will be working  with Tag helpers many times in the application.

53:18

But to just give you a brief overview, you can see  in the first tab here we have HTML helpers, and

53:25

tag helpers simplifies all of them, because we can  use the existing label tag. And we will just add

53:32

a Tag Helper ESP four, and we will find that to a  model. We have same Tag Helper right here as well.

53:41

Then in the last example, here I am displaying tag  helpers where previously we were using HTML dot

53:48

begin form. And now within the same form tag, we  have the tag helpers of where it should be posted.

53:56

So you can see things are getting much more  simpler when we are using Tag helpers. That

54:02

being said, if you're coming to dotnet, brand  new, and you have never worked with HTML helpers,

54:08

this is even better. You do not  have to worry about the old syntax   and what is different. We will be working  with Tag helpers in the upcoming videos.

54:19

But I just wanted to give you a brief snippet  of what tag helpers are and how they look like.

54:26

When we are using the same tags, we just use  the ASP hyphen and then the Tag Helper name.

54:33

That being said, we will explore all of this  in much more details in the upcoming videos.

Action Result

54:41

When you will be working with  a dotnet core application,   you will see the return type of action  result. And it does not matter if you're

54:49

using an MVC application or a razor page  application. In both the cases you can see

54:56

we have eye action result in MVC We are  returning back view in this example. And in

55:03

razor PAGE PAGE handler, we are returning back to  the page. But the return type is I action result.

55:12

I action result is a generic type that implements  all of the other return types. Now, if you want to

55:19

be explicit about the return type in both of these  cases, then that would look something like this.

55:26

If the return type is view, you can write View  Result. But in razor pages, when we return back

55:32

to page, you can write page result. So now what  is the advantage of action result? So first,

55:41

let's understand some theory. So action result is  a pattern class for many of the derived classes

55:49

that have associated helpers. The eye action  result return type is appropriate when multiple

55:56

action results return type are possible in  an action. Let's take a look at some of the

56:03

helpers and action wizard. So first, let's  take a look at what is there in razor pages.

56:10

A razor page can return content file not found  page a partial result, and redirect to different

56:18

pages. For all of them, we have returned types  like content result, file content result,

56:25

not found result, page result, partial result, and  so on. If you are working with an MVC application,

56:33

we can return back views, partial views, we can  redirect to action, return JSON and so on. So

56:41

if you are returning any one of these, then you  can use the individual return types for result

56:48

based on the helper method. But what if  you were returning something like this?

56:54

Right here in MVC, you can see we are returning if  true, let's imagine that was some condition here.

57:01

Based on that if that condition is  true, you want to redirect to action,

57:06

else you want to return to view this time you  cannot have two different return types. If you

57:12

use View Result here, return view is working,  but redirect to action will throw an error.

57:19

Similarly, if you use redirect to action result,  then redirect to action works. But return view

57:27

will give you better if you are working on  razor pages and you have the same situation.

57:33

If you use page result, and return page will  work. But redirect to page will fail. And if

57:40

you use redirect to page result, then return page  will fail. So what is the solution to all of this,

57:49

the solution is to use I action result in both  places. Because this is a parent class. So it

57:56

does not care which of its implementation is being  returned, it will be able to handle all of them.

58:03

So that is a brief overview on how action  wizard will help us with the return type

58:09

from action method in MVC application or  page handlers in a razor page application.

Hot Reload

58:19

Now that the project has been created, there  are few files and folder structures right

58:24

here. We will go into all of this files and folder  structure in the upcoming videos. But right now,

58:32

inside the views folder, we have home. And inside  there, we have index.cs. HTML. This is the default

58:41

home page of our application. So if I open this  up, you will see there is some welcome text here

58:48

and a paragraph, which is inside a div tag. Let's  run our application and see what is the output.

58:58

It loads up the default page of the website  where we have the project name here,

59:03

a homepage and a privacy page. You can  see on the homepage, we have some text

59:09

right here. And that HTML is exactly same as what  we see inside the index.cs. HTML. I do not want to

59:19

go into much technical details. But let's say  you are making some changes inside your view.

59:25

If you save that, you go back to the application.  And if you refresh here, you're directly see the

59:32

changes right here. If you notice in Visual Studio  2022 There is this icon, which is for hot reload.

59:41

This is a great capability that has been added  with dotnet sinks. If this does not refresh

59:48

and reload for you. You can go here and make  sure the hot reload on File Save is enabled.

59:54

What that will do is whenever you make some  change here and you save if you go back refresh,

1:00:01

you can see the change is  being reflected in the UI.   This is very helpful when you are designing  something with MVC or razor application.

1:00:13

But if you notice the first time, you had  to manually come to the page and refresh,   and after that the hot reload automatically works.  So I wanted to walk you through that advantage.

1:00:25

Now, if you were working with an application,  which is not in dotnet, six, then you will have

1:00:31

to right click on the project, and you will have  to add a new get package by clicking the Manage

1:00:37

nougat package. You will search for a package  runtime compilation. And you can see we have

1:00:44

Microsoft that ASP NET Core dot MVC dot razor  dot one time this package was needed before,

1:00:52

so you can hit the Install button to add that to  our project. Even though that is not needed here,

1:01:00

I still want to add that then you will  have to go to program.cs. And right here,

1:01:06

we only have ADD controllers with view. When you  add razor pages, you will add builder dot services

1:01:13

dot add razor pages. And on there you will have  to enable the Add razor runtime compilation.

1:01:21

But because of hot reload and the magic  that we have, this is no longer required.

1:01:28

So in the future videos, when we will be adding  the razor pages, I also add the razor runtime

1:01:34

compilation, because when the video was originally  recorded in dotnet, six, hot reload was still in

1:01:41

testing. So that's why I have added the razor  one time compilation down the road. But you can

1:01:48

skip that and you will just add the builder dot  services dot add razor pages. So that's a brief

1:01:56

overview that I wanted to give you with the heart  reload in dotnet six. That being said in the next

1:02:03

video, let's first examine all the files and  folder structures that have been created here.

Create Category Model

1:02:12

When we work with any project, the main piece  or the heart of the application is the data.

1:02:18

And we need a database to store our data. So we  will be using SQL Server to create a database

1:02:26

and store all the data for our website. With  dotnet core, you might have heard about term

1:02:32

condors Entity Framework core as well. If not,  do not worry. But Entity Framework core is what

1:02:39

we'll be using to create database and we will be  using the same to perform all the data operations.

1:02:47

So with dotnet core, you do not need stored  procedures or writing SQL statements in the

1:02:52

code. Entity Framework is a spot editor that will  help us with all the data related operations.

1:03:00

That makes sense, it does not mean that you cannot  use stored procedures, you can still use stored

1:03:06

procedures if you want. But typically with MVC  application Entity Framework core is used to

1:03:13

manipulate the data layer. We will go into those  details in the upcoming videos. But right now,

1:03:20

we first need to create our model model will  basically resemble a table in database. It is

1:03:27

not always the case. But whatever tables you have  in your database, you will need a corresponding

1:03:33

model for the code first migration. So inside the  Models folder, we only have added view model. Let

1:03:42

me create one more model here, we will right  click there. And we will add a class file.

1:03:49

We want to create a table for category. So we will  call our model as category. And let's add that

1:03:58

it creates a public class file and it places it  inside the namespace of bulky book web doc models.

1:04:05

That is just the location so it knows that it  is inside bulky book web Models folder. That

1:04:11

is the category class right here. Now inside  this category class, we need to create all the

1:04:18

properties that we found. For our table. We  want to create ID name and a display order.

1:04:27

So to start we can write prop and it is a code  snippet. So once your type prop and hit tab two

1:04:33

times it will automatically create a property  the ID column we will keep that as integer

1:04:41

and I will call that as ID. Next property that we  want to create will be the name of the category

1:04:49

that will be a string. So we will create that  we press prop again we want to display order.

1:04:56

Now display order will be an integer so Let me  add that. On top of that, let's say for logging

1:05:04

purpose, we want to create a date time property,  which will have the date on when this category was

1:05:10

created. So we will call this created date time.  Now the created date time, let's say we want to

1:05:18

set a default value to that. So in order to do  that we can use equal to sign. And we can just

1:05:24

assign the date time dot now like this. That way,  the default value will automatically be assigned

1:05:32

to the created date time when we create an object  of this class. So perfect, we have created our

1:05:39

model. And then we want to push this model to our  database to create a table with this four columns.

1:05:48

But when we create a database table, we might want  to add some more configuration. Like we might want

1:05:54

to see that id is a primary key. And since it's an  integer, we want to make that an identity column.

1:06:02

So we do not have to populate that when we create  a row inside the new table that we will create.

1:06:08

We could also add a validation that name  is a required property. And it should not

1:06:14

be now if we had to write a SQL statement,  we could have done all of that by using not

1:06:21

an identity in our SQL script. But how  will we do that using Entity Framework

1:06:28

for that dotnet team has come up with alternatives  called as data annotation. So on the property

1:06:35

where you want to configure some more details,  we have special attributes that you can use.

1:06:42

Let me walk you through them in the next video.  Now we want to configure some of the requirements

Add Data Annotations

1:06:50

that we have in a SQL script. Like we want to make  this ID column, an identity column, which will

1:06:56

be the primary key of this table. In order to do  that, we have a data annotation, or an attribute

1:07:04

known as key. If you enter that, you notice  the red squiggly lines. If you hover on that,

1:07:11

it will display an error, that key attribute  could not be found. That is because we have

1:07:17

to add a USING statement. So you need to make sure  your keyboard is on the key. And then if you press

1:07:25

control.it will display that you can add USING  statement to resolve this. So we will press enter,

1:07:32

that will add the USING statement which  has the data annotation that we want.

1:07:38

Once this key annotation has been applied,  it will tell Entity Framework core that hey,

1:07:43

when you create a script to create this table,  you need to make sure that it is a primary key.

1:07:50

And it should also be an identity column. It  does all the configuration and talking by itself.

1:07:57

We just need to write one attribute and everything  will be done for us. Pretty simple, right.

1:08:04

The next thing that we wanted to do is we want  to make sure that name is a required property.

1:08:10

So here we have another attribute, which  is required. Once you assign that when

1:08:17

it creates the script, it will make sure  that name is not an allowable property.

1:08:23

Now we have talked about these two, but there are  more annotations that you can do for validation.

1:08:30

With this in place, the initial equation of  our model looks good. How do we create that

1:08:35

in our database. Now, let's take  a look at that in the next video.

Add Connection String

1:08:43

have added the data and rotations 10 We have  created our model, we need to create a table and

1:08:49

a database inside SQL Server. So let me open up  SQL Server Management Studio. Make sure you have

1:08:57

that installed on your machine because that is  what we will be using to play with the database.

1:09:04

Once the SQL Server Management Studio opens up,  we will be using our local database. Now for some

1:09:11

of the users it could be local TP backward slash  MSSQL local DB. For others, it could be just a dot

1:09:20

I have both of the server name configured. So  I will use any one of them and I will hit the

1:09:26

connect button. As long as it is able to connect  that server name is what you will be using inside

1:09:33

your connection string. If you use something else,  then it won't work. So I will be using dot. But on

1:09:41

your end you need to check if dot works. You  can use that or the local DB MS SQL local DB

1:09:48

should also work. Let's hit cancel here. And if  we expand the databases, you can see we have few

1:09:55

database here but I want to create a new database  for Project. So for that, we have to create a

1:10:03

connection string inside our project. In order  to store all of the secrets of your application,

1:10:11

you will be doing that inside the app settings dot  JSON file. You can hard code the connection string

1:10:18

inside the class file. That is a bad approach.  App Settings dot json file is the file where

1:10:24

you should have all the secrets. That way, if you  have to update anything in the future, you know,

1:10:30

it will be available inside the app settings dot  JSON file. And then you can create different app

1:10:37

settings for different environment. Like if you  have development, staging, preview production,

1:10:43

you can create all of those app settings. And  you can configure to use that app settings

1:10:48

when you deploy the application. That way based on  your environment name, you can use different app

1:10:56

settings, or different database and so on, we will  be working directly in App Settings dot JSON. And

1:11:03

that should work with the local development. So  this is a simple JSON file. Now we'll be add a new

1:11:11

connection string in here. As you can see, it is  just a dictionary with string key and value pairs.

1:11:19

What you have inside logging is known as  a block, where inside the locking block,   we have another block of log level. And that  has three key value pairs. So we can either

1:11:30

create a block, or we can directly configure the  connection string. Now the dotnet team thought

1:11:37

that connection string is very common, and almost  all of the projects will have a connection string.

1:11:43

That is why they have created a default block for  connection string that we can use. If we want,

1:11:50

you can use something else. But you can also use  the default block name that they have given. If

1:11:57

you see it is automatically suggesting me that  block name, which has connection strings, we can

1:12:03

use something else. But I want you guys to use the  exact same name for now. And I will also tell you

1:12:10

the reason behind that in the upcoming videos. So  inside this block, we will have a key value name.

1:12:18

So that key name, you can use whatever you want, I  will just use default connection, you can use any

1:12:26

name that you want for the key right here. And  we need to paste our enter a connection string

1:12:32

within the double quotes on the right side. So  the first parameter here will be the server name,

1:12:39

I am using.so That is what I will write  if you are using a different server name,

1:12:45

like if we go back to SQL Server. And if you  were using the MS SQL local DB to connect,

1:12:51

I want you guys to copy this, go back to the  application and paste that here. I am using

1:12:58

the local server with a card. So that is what I  will use. And then we need the next parameter.

1:13:04

If you have to separate anything in a connection  string, we will use a semi colon. The next

1:13:10

property that we have is the database name. Let's  call this bulky. Then after that, we want to set

1:13:17

the trusted connection flag to be true. Make sure  there is no spelling mistake here. And that is all

1:13:24

that we want to configure in the connection  string. You can make it more complicated,

1:13:29

but we will keep things pretty simple. We have a  database name that we want to create inside this

1:13:36

particular server that we are able to connect in  SQL Server and the cluster connection is true.

1:13:43

That way when we connect here, we  can just use Windows authentication.   If you have a user ID and password, then you can  use the different settings in connection string,

1:13:54

I will be using the default crusted connection.  So with this we are using a special block that is

1:14:01

available with the name of connection string.  In there we have a key of default connection

1:14:06

and value has the connection string. Now how can  we use this connection string to actually create

1:14:13

the database and create our category table inside  the database. Let's do that in the next video

Add ApplicationDbContext

1:14:23

we need to tell our application that we  will be using SQL server and you have to

1:14:28

use this connection string to establish the  connection with SQL Server. For that we will

1:14:35

be using Entity Framework core and we will  have to create an object of the DB context.

1:14:42

Using that DB context we will be able to make  connection to the database. So how do we do that?

1:14:50

It is best to create a new folder for all  the data related changes. So here let me

1:14:56

create a new folder with the name of data  In SEC there, we will create our DB context,

1:15:04

we will create a class here. And I will call that  application DB context. You can use any name that

1:15:13

you want here, but I'm using DB context so that  it's easy to identify. Let's add that class file.

1:15:21

Now we need to inherit this class file from the  DB context that is inside Entity Framework core.

1:15:28

In our project, we have not added Entity Framework  core right now. So there are two ways to add that

1:15:36

we know that this will inherit from the  DB context. So we can type that and you

1:15:41

will see the red squiggly lines on there, if  you press Control dot inside the suggestions,

1:15:48

it will display that you can install a  package Microsoft dot Entity Framework core

1:15:54

that will automatically find an install the latest  version. If you want to do that yourself, you can

1:16:01

right click on the project. And you can select  Manage NuGet packages, or you can go to tools,

1:16:08

we have new get package manager, and we can  open up manage nougat packages for solution.

1:16:16

Here, you will have to go to the Browse tab to  see all the new get packages that are available.

1:16:22

Now right now I'm using dotnet  six with the preview version.   So I have checked the include pre release  right here. The package that we are looking

1:16:32

for is Entity Framework core. Let's press Enter.  And the first package is Microsoft dot Entity

1:16:40

Framework core. We are using the previous seven.  So let me select that. And we will install that.

1:16:52

Once that is installed, if we go  back to our application DB context,   and now if we press Control dot here, it will tell  us that we just have to add the USING statement,

1:17:03

since we have already installed the package.  So we've add the USING statement for our DB

1:17:10

context. Once we do that, then there is one line  of configuration that we have to do inside the

1:17:17

constructor of this class file. You can think  of that as the general syntax that is needed

1:17:23

to establish the connection with Entity Framework.  So first, we need to create a constructor,

1:17:30

you can type C, D or R and press tab twice. You  can see it is a code snippet for constructor.

1:17:37

So once you press tab twice, it should  automatically create the constructor, we just

1:17:43

have to write some parameters here. Because when  we get the DB context, we need to pass that on

1:17:49

to the base class, which is dB context. So here,  we will have to configure the DB context options

1:17:58

on the class that we are on right now, which is  application DB context, we can paste that here.

1:18:05

And I will call this options. So here we  are saying that in the constructor here,

1:18:11

we will receive some options. And those  options, we just have to pass to the base class,

1:18:17

which is dB context. This is a general setup that  you have to do that will configure our DB context.

1:18:26

Now once you configure our DB context, we still  have one main feature, we still have to create

1:18:34

our category table inside the database.  So whatever models that you have to create

1:18:40

inside the database, you will have to create  a DB set inside the application DB context,

1:18:47

the file that we are currently working on. Now  how do you create a DB set that is pretty simple,

1:18:55

we will say public DB set here, and we need  to write the model name. That model name is

1:19:02

category. And once you write that, you can see the  red squiggly lines. That is because it cannot find

1:19:09

anything with the name of category in the same  file, we need to add the USING statement to tell

1:19:15

it that category is inside the Models folder. So  if you press ctrl.we have that USING statement.

1:19:23

Let's add that and the next parameter here is the  table name. So if you call this categories then

1:19:31

inside database the table that will be created  will be called as categories and not category.

1:19:38

So that looks good. And we will add the getter  and setter. That's all that we had to do

1:19:44

to create the category table. What this will do  is it will create a category table with the name

1:19:51

of categories and it will have four columns  that we have wrote inside the category model

1:19:59

when it creates stack table it will make sure that  ID is an identity column and name is a required

1:20:05

field as well. So, you can see it is doing all the  configuration by writing just few lines of code.

1:20:13

Now, when you are working with Entity Framework  core, there are two models, one is code first

1:20:19

and one is database first what we are doing is  code first, because here we are writing the code

1:20:26

of our model and based on that model, we will  be creating the database. So, that is the code

1:20:32

first approach database first approach will be  something right database is already created. And

1:20:39

based off that database, you will be scaffolding  models. I personally am a big fan of code first.

1:20:46

And that is what I have been using in all of my  production application. Because I do not come

1:20:52

from a DBA management point of view, I am more of  a full stack developer who works with the code.

1:20:58

So if Entity framework code will manage database  for us, I will be very happy with that approach.

1:21:05

So with that we have added our  DB set for the category table.   But we are missing one small configuration or  application still does not know that it has

1:21:16

to use the connection string that we wrote  in app settings. And it still does not know

1:21:22

that it has to use Application DB context to  create a DB context and that it has to work

1:21:29

with SQL Server. So let's see how we can  pull everything together. In the next video,

Setup Program.cs to use DbContext

1:21:37

we just need to tell our application that it  has to use the DB context which has inside

1:21:42

application DB context. And then it has to use a  SQL Server using the connection string that were

1:21:49

defined inside the app settings dot JSON. We will  tell our application to do that inside program.cs

1:21:58

where we configure the services that our  application will use. So here we have the comments

1:22:04

to add the services to the container make sure  you always do that before you build the builder.

1:22:11

So right here, we want to add a new service. So on  builder dot services, the surface that we want to

1:22:19

add is dB context. So you can see we have dot  add DB context and that expects a class file

1:22:28

the class file that we are using for DB context  is the application DB context. If you press

1:22:35

control.we will just have to add the  USING statement to the Data folder.   Now, when we configure this DB context, if  we go back to the application DB context,

1:22:46

right here, we are passing the options and we  are sending those options to the base class. So

1:22:54

inside options, we have to configure use of SQL  Server and connection string. So here we will see

1:23:01

options goes to this is just the syntax, we will  say options dot that is a method with the name of

1:23:10

use SQL Server, but it will not be available like  that, even if you press control.it will not give

1:23:18

you the package name that you have to add. So for  that, we will have to go back to the Manage nougat

1:23:25

packages. And we will have to install a package  which is SQL Server with Entity Framework core,

1:23:33

which is Microsoft dot Entity Framework core  dot SQL Server. Make sure you are using the

1:23:39

consistent version. If one version of the package  you are using is preview seven and other one is

1:23:46

preview five, or even dotnet five, and things  won't work and you will run into error message.

1:23:53

So always make sure that you  are using the same version.

1:23:59

Let me install the SQL Server and we will close  the nougat packages. Now if you press ctrl dot,

1:24:07

you will see the USING statement using Microsoft  dot Entity Framework core. We will add that and

1:24:14

on the SQL Server we have to write the connection  string. So where exactly is our connection string.

1:24:22

That is inside app settings, we used a special  block with the name of connection strings.

1:24:29

Since we put our connection string inside this  special block theme that I was talking about,

1:24:34

we can directly use the key value  here to extract the connection string.

1:24:40

Let me copy this name. We will go  back to our program.cs and right

1:24:46

here on builder dot configuration, we have an  existing method that is provided which is get

1:24:52

connection string. Inside this method we have to  pass the string name which is default connection

1:25:00

Once you use that, it will automatically find the  connection string and configure our SQL Server.

1:25:08

Now this cat connection string is a  special method. And this method will   only look for this default connection inside  a block with the name of connection strings.

1:25:19

If you named this connection strings one, then it  will not be able to find this connection string,

1:25:25

because this method will only look inside  the block, which is connection strings.

1:25:31

So that is why I said to name it exactly the  same, if you wanted to name it something else,

1:25:37

we have different ways of getting that. But I  do not want to go into those details right now.

1:25:44

So with this, our DB context will be  configured with the connection string.

1:25:50

So all the configuration is done that was needed  for the DB context. Now we are on the final step,

1:25:57

where we have to create database, and then the  table inside SQL Server. Let's take a look at that

1:26:05

in the next video. Now that the  program.cs has been configured,

Check Database

1:26:11

the next step that we have is to actually  create the database and the table.

1:26:16

When you are using Entity framework code.  First, there are migrations that you have   to run using Entity Framework to push the changes  to database. It is not as complex as it sounds,

1:26:29

we have done all the setup that is needed. In  order to run the migration, you will go to Tools

1:26:36

NuGet package manager, and this time, you will  select the package manager console. The first

1:26:43

thing that you have to do is you have to add  a migration. Migration is basically keeping a

1:26:49

track of all the DB changes that are needed. And  once that migration is created, you push that

1:26:55

migration to the database to actually create  the database or make changes to your table.

1:27:02

Let's see what that is. The command to add a  migration is add hyphen, migration. And then we

1:27:09

have to give our migration a meaningful name. Do  not use any spaces when you are writing the name.

1:27:16

The name that I want to give to our migration is  Add category to database. Let's hit the Add button

1:27:24

and we see an error message. The error is because  when you run add migration, you have to add a new

1:27:31

get package. You see the term add migration is  not recognized. I want you guys to copy this error

1:27:39

and try to find out what is the package that is  missing from Google. I can give you the answer

1:27:45

directly. But I want you guys to Google what  package is required based on this error message.

1:27:52

If you just copy that error and paste it here, you  will see the very first and if you scroll down,

1:27:58

you will see the package name. It is so simple.  Just install Microsoft dot Entity Framework core

1:28:04

dot tools. That is the package that is required  to enable migrations in your project. Whenever

1:28:12

you face any error, it is best to just Google  that error. That way you can find the solution

1:28:18

much easily and much quicker. So let's go to  our project manage nougat packages, we will

1:28:25

paste the package name, and we will install  the same version that we have in our project.

1:28:34

Perfect that is done, we can close this, I can  open up the package manager console again. And

1:28:40

I will run the same command one more time.  This time it should complete successfully.

1:28:49

And if you notice on the right hand side here, a  new folder with the name of migrations is created.

1:28:55

And there are two files that got added. Our  migration file is the first one and that is opened

1:29:02

up on the screen as well. Inside migrations, we  have two methods, one is up and one is down. The

1:29:10

up method is what needs to happen inside the  migration and down is if something goes down,

1:29:17

we need to rollback the changes. So do  not worry about the download right now.

1:29:22

Let's see what's happening inside the app  method. On the migration builder, we have

1:29:28

a method with the name of CREATE TABLE and it is  creating that table with the name of categories.

1:29:36

That is the exact name that we defined right  here. So it will create a table with the name

1:29:41

of category and then that will have columns. The  first column will be the ID column which will be

1:29:49

of the type integer and nullable is false. Since  we said it is a key column, it is automatically

1:29:56

making that an identity column and increment  that one every time. Pretty smart, right?

1:30:04

The next thing is the name column. And you  can see the nullable is false here. If you

1:30:10

did not add the required property, that nullable  would be true, you will say that display order

1:30:18

is also a nullable false, but we did not add that  inside category. And the reason behind that is

1:30:24

it is an integer property and not a string.  That is why nullable is false, because it's

1:30:30

an integer. Last, we have the created date time  of the type date time too. And that looks good.

1:30:38

If you see on the constraints, it is also adding  a primary key on the ID column with a name. So

1:30:45

migrations is exactly what we wanted. But can  we take a look at the sequel that gets executed?

1:30:52

Now, what Entity framework code does is  based on this migrations, it will create

1:30:58

an optimized version of the SQL query. And it  will automatically run that on the database,

1:31:05

you do not have to do anything with that what  you work with is just the models, you create a

1:31:11

migration and you push that to the database. So  once you verify that migration is looking good,

1:31:18

you can just run the command update database.  And that will push the migrations to database.

1:31:26

We have an error here, let me go to app settings.  And of course, we do, this will not be colon,

1:31:34

this will be equal to right here, and they  are separated by a semicolon. That was my bad,

1:31:41

we will go back to the package manager. And you  can also see the error said server and colon was

1:31:47

not supported, because it should be equal  to let's update the database one more time.

1:31:54

Perfect the migrations were completed. But what  actually happened is first it will connect to the

1:32:00

server, it will check is there a database with the  name of palki there won't be anything right now.

1:32:07

So it will create that database first. And then  the migrations here, it will convert them to SQL

1:32:13

and execute them on our database. So if we go  back, now, if we refresh the database, you will

1:32:20

see the new database with the name of bulky and  if you open the tables, that will be two tables.

1:32:29

Let's examine the columns  inside the Categories table.   It should have four columns. And that looks  great. Now what is inside the EF migrations,

1:32:40

if you do the Select Top 1000 entity framework  code keeps a track of which migrations have been

1:32:46

applied. So next time when you run the update  database, it will not apply the same migration,

1:32:52

it will only apply the migrations that  have not been applied on this database.

1:32:58

Entity framework code is pretty smart with all  the configuration and cracking that it needs to

1:33:03

do. So with that, using Entity Framework core, we  have created our database. And we have added our

1:33:10

table based on the model inside our main project.  So with that, let me continue in the next we do.

Create Category Controller

1:33:21

Now before we work on anything  else, that we run the project.   Right now it is running on port 5001. Let me  change that so that the command prompt is not

1:33:32

opened every time, we can just run that using  IIS profile by clicking here. That way it won't

1:33:39

run on the port 5001 It will get the new port that  is defined in the project config, I'm getting an

1:33:47

error that I'll have to open Visual Studio under  an admin account. So let me do that real quick.

1:33:55

With it open the project again. We have the IIS  Express here. Let's try to run it this time.

1:34:04

And perfect. So we have home and privacy but both  of these pages are inside the home controller.

1:34:11

Now in category we will be creating editing  and deleting category. So for that rather

1:34:18

than working on the home controller, let's  create a new controller for our category.

1:34:24

So inside the controllers folder, we will  right click Add a controller and we have

1:34:30

few options here. We will go with the empty  controller to keep things very simple and start

1:34:36

from scratch. I will call that as category  controller. Now when you name a controller,

1:34:43

make sure that you append controller at the end.  That is a required field. Whatever name you want,

1:34:50

must come before the controller. So once you  have that name, we will click the Add button.

1:34:57

For effect we get an empty control With an  index action method, our home controller

1:35:04

also had the index action method. And our category  controller also has the index action method. But

1:35:12

the index action method in home controller  has a view that is inside the home folder.

1:35:18

Our category action method does not have a view  right now, you can add the view in two ways.

1:35:25

First, you can create a folder with the  name of controller, which is category   inside the views. And then you can add index  view inside there. Or if you want to do that

1:35:36

directly from the controller, you can just  right click on the action and you have ADD view,

1:35:43

we will be selecting the razor view here. Let's  hit the Add button. Now when you add a razor view,

1:35:50

there are a few configuration that you have to  do. First, what will be the name of the view,

1:35:56

we will give that the same name as the action  method, which is index, then do you want to use

1:36:03

any templates when you create this view? If you  hover here, you can create a view for create

1:36:09

delete details, edit and list. I do not want to  go into those details right now. But we will come

1:36:16

back to that later on. If you change that, then  model and data context class will be enabled.

1:36:23

But we don't want to focus on that right  now. We want to start scratch with an empty

1:36:28

template. Next is a partial view. Partial view  is basically like user controls in the platform.

1:36:37

So it will be rendered inside some other view. So  if you select a partial view, the layout page will

1:36:43

not be used. Because you do not need any master  page if you create a user component, because you

1:36:50

will be calling that inside some other view. We  don't want to take a look at that right now. Let's

1:36:57

keep it simple. Next, we will use the layout page,  we want our view to use the default master page.

1:37:05

So if you keep it blank, it will by default use  the layout that we have set inside the view start,

1:37:11

which is underscored layout, we want it to be  consistent. So that is what we will be using.

1:37:17

And we will hit the Add button. It will scaffold  few dependencies. And then it will create the view

1:37:25

if you had any errors while building the project.  And if you try to scaffold that will fail,

1:37:31

it will ask you to resolve the errors. And only  Once your project is building successfully, you

1:37:37

will be able to scaffold the views. So perfect.  Our view has been created. And let me call this

1:37:44

index category. With that, let's continue. In the  next video. We want to display all the categories

Retrieve all Categories

1:37:53

if there are any in the table format, inside  the index view. For that, we will have to

1:38:00

actually retrieve all the category list from our  database inside the index action method. So that

1:38:07

we open up the layout page right here, where we  have two options of index and the privacy page.

1:38:15

Rather than privacy. Let me display the category  right here. And in order to go to the Index page,

1:38:22

how can we navigate to this particular view that  was created, it is inside category controller, and

1:38:29

we have index action method. So here to navigate,  we can define the controller name is category.

1:38:37

And the action method is index. We are not  using areas here. So you can keep that blank.

1:38:45

Or you can also remove them if you want. If you  save this, let's run the project one more time,

1:38:52

and make sure we are able to navigate to our  action method. So we have category when we click

1:38:59

there, perfect, we see the index categories. That  is what we have inside the view. So we are able

1:39:06

to navigate there successfully. Now we need to  retrieve all of the categories from our database.

1:39:13

So let me switch back to database. And just for  Tommy purpose, let me edit the top 200. And let

1:39:20

me add something here. So I have just created  one dummy record inside the table. And right now,

1:39:29

I have created one dummy record inside the table.  Let's go back and we want to retrieve that inside

1:39:37

the category controller. Now I have said before  that we will be using Entity Framework core.

1:39:43

And the main file that we have is the application  DB context. Using that we can access our database

1:39:50

object. Because inside that file, we have  the Categories table. So how can we create

1:39:57

an object of this application DB context?  and use that to call our database and table.

1:40:04

That is the beauty of dependency injection, we  do not have to create an object of this class,

1:40:11

everything will be done for us, because we  have configured that inside our container,

1:40:17

we want to use this service. So because  of dependency injection, we will not have

1:40:23

to create its object, we can think that the  object is already there, we just have to tell

1:40:29

application that please send me the object of  application DB context. How do you request that

1:40:36

inside the controller, you want the application  DB context to work with database. So I will first

1:40:44

create a private read only field,  which will be application DB context.

1:40:50

And we'll call that underscore dB. We will  add the USING statement. And great. Now we

1:40:57

need to tell our application that we need an  implementation of this application DB context

1:41:03

where the connection to database is already made.  And I can recreate some records right away. So for

1:41:10

that, you will have to use constructor see to  our tab tab. And here, whatever is registered

1:41:17

inside the container, you can access that inside  our container, we have registered application DB

1:41:24

context. So inside the constructor, we can  get an implementation of that, just like this.

1:41:32

So this DB will have all the implementation of  connection strings and tables that are needed to

1:41:38

retrieve the data. So we will populate our local  DB object with this implementation. So we'll say

1:41:45

underscore db is equal to dB. And then we can use  this underscore db to retrieve our categories. The

1:41:54

syntax of that is very simple, we will create  a bar let me call this obj category list,

1:42:03

we will first access the underscore dB. And on  there, we will have all the DB set. The DB set

1:42:10

that we want to work on right now is categories.  And we want to convert it to a list and retrieve

1:42:16

that. So it will go to database, it will retrieve  all of the categories, it will convert that

1:42:23

to a list, and it will assign that inside the  category list. You can see how beautiful this is,

1:42:30

you do not have to write a select statement  to retrieve all the categories from the table.

1:42:35

There is no SQL coding required. You just have to  write underscore DB dot the DB set name. And that

1:42:43

will retrieve all the records. You convert that  to a list and you assign that to a variable. Now

1:42:50

how do we test that this is working, we will  add a debugging point by going to this pan,

1:42:56

you see the red dots. If you click here, we are  setting a debugging port. And then if you run

1:43:02

IIS Express here, your application will hit  the debugging point when we load that page.

1:43:09

So if we click on category, it will hit a  debugging point. You can see that is yellow

1:43:15

now. And if you hover on the category  list, you can see there is one count.

1:43:21

If you expand it more, you can see it is the  same record that we created inside the database.

1:43:28

So it goes to the database, it fetches the  categories. And it does that automatically.

1:43:34

We have a list that is already populated here,  we just need to pass this list to our view.

1:43:41

And we can run a for each loop on that  list. And we can display all the categories.

1:43:47

Let's do that in the next video. Let me remove the  breakpoint. And let's do that in the next video.

Display all Categories

1:43:56

Know that we have all the categories  inside this particular object,   we can copy that and we will pass that to our  view. And I do not like using variables here.

1:44:07

I am a fan of using the strongly typed  so this will be i enumerable of category

1:44:16

we'll have to press ctrl.to Add the USING  statement. Now since we have an IEnumerable

1:44:23

here and not a list, if you want, you can also  remove the.to list. That is not a required thing.

1:44:30

But if this was a list, then you would have used  the.to list. So we are passing that to our view.

1:44:39

Now that we are passing an  IEnumerable of category to our view,   we need to capture that inside our view as well.  So let me open up that view I have right here.

1:44:51

But if you do not have that open, you can go to  category and index view. So right here, we first

1:44:58

need to capture the model that we Passing from the  controller, for that you will be using activate

1:45:04

sign, and you will write model in all lowercase.  If you make this a capital model that won't work,

1:45:12

in order to fetch what is being passed from  the view, you have to use all lowercase at the

1:45:18

rate model sign. So what exactly is being  passed from the controller action method,

1:45:24

it is an IEnumerable off category, we can copy  this, and we can paste that right here. That way,

1:45:32

our view knows that whatever we receive here  will be an IEnumerable of category. Since it

1:45:39

is I enumerable, we can iterate through that.  Now if you have never worked with razor pages,

1:45:46

you can use both HTML as well as some C sharp  code directly inside the view. So first, let's

1:45:54

use the table attribute. Give that some classes  of Table Table bordered to give it a border.

1:46:02

And we'll say table striped, we can  also give it a style of width 100%.

1:46:13

Inside the table, we have tea head and tea body.  Let me add both of them. So tea head will be for

1:46:22

headers. So we will have a TR tag in there and  a th tag, what will be all the columns that we

1:46:29

have to display. The first one will be category  name, and the next one will be display order.

1:46:38

You can also display the date time and ID if you  want, like we just want to display two columns.

1:46:45

That is the header. Now inside the body,   we will need a for each loop to iterate through  the I enumerable and display the actual values. As

1:46:56

I said before, we can use the shortcode directly  inside the views using the Add the rate directive.

1:47:03

So if you press activate, you will see we have  a for each right here. Syntax is pretty simple,

1:47:11

we will create a wire here we can call that  Obj. And that is inside our model of the page.

1:47:19

So on top we use the lowercase model. But here  model will have the first letter as capital.

1:47:26

So you can see how it is different. That way it  will iterate through all of the properties inside

1:47:33

this model in a for each loop. Now for each of the  property, we want to display a table row. So we

1:47:40

will add TR here in that we want the TD tag. First  we want to display the name. So again, that name

1:47:48

is inside this C sharp variable object. So we will  use the activate sign on we will say obj dot name,

1:47:57

I can give it a width of 50% here. And we can  copy this and paste it one more time. Then we

1:48:07

have display order. So we will say obj dot display  order. Let's give it a width of 30%. This looks

1:48:15

great. Let me save this and run the application.  If we go on category now, you will see our one

1:48:24

category that we have inside the database. This is  looking perfect to display all of the categories.

1:48:32

Next, what I want is Edit and Delete buttons here.  And I want to make this a little bit more pretty.

1:48:39

Before that I will be using boots watch themes.  And let me show you that in the next video

Bootswatch Theme

1:48:46

loves a pretty application. Because of that I  want to use boots watch.com, which has free themes

1:48:54

for Bootstrap. If I scroll down, I want  to use the solar theme for our project.

1:49:02

We can hit the download button. And let's  download the bootstrap dot css. Let's open

1:49:08

that file and it is using Bootstrap five. That's  perfect. Let's copy this complete Bootstrap.

1:49:16

And let me stop the application site WW root where  we have CSS or Bootstrap is inside the lib folder.

1:49:24

But here let me just create  a new item style sheet.

1:49:32

We'll call that boot swatch team. And let  me add that I will paste the theme that we

1:49:38

copied and we will be using this in our  project. Looks good. Let me save this.

1:49:45

Another change I would do is inside site dot  css. They have added a btn primary. So let

1:49:52

me remove that and the anchor tag. We will be  using what is there inside our boots watch theme

1:50:00

In order to add this to our project, we  will have to go to underscored layout.

1:50:05

And we should have our CSS at the very top. This  time, we do not want the default bootstrap theme.

1:50:14

We want to use our boots watch theme dot css. Now  since we are using Bootstrap five with boot spot,

1:50:22

if we go to bootstrap.com. Let's click on getting  started in production, we are in Bootstrap five,

1:50:32

let me copy the je s bundle that we have here.  We will copied that and we will go back to our

1:50:39

project inside the underscored layout that  may replace the GS bundle that we have,

1:50:45

with the one that we copied. As of this recording,  the default version of Bootstrap is not five in

1:50:53

dotnet, six, this might change in future. But that  is the reason I'm switching to the latest version

1:51:00

of Bootstrap. With that change, let's run the  application and see if something is different.

1:51:09

Right, you can see a dark theme has been  applied. So things are getting different.

1:51:15

Let me switch back to the boot swatch. And let  me go back to the theme that we have. Let's click

1:51:22

on Preview. Here we have a nav bar, let's use  that. I will copy everything that we have here.

1:51:31

Let's go back to our application. And I will paste  that right here before the nav bar that we have.

1:51:39

Now since we are making changes inside the view,  if you remember, when we created the project,

1:51:45

I enabled the razor runtime compilation. So  what that should do is when we save here,

1:51:52

and if we go back to our application and  refresh, it should load the changes, because

1:51:57

we are making changes directly inside the view.  But that does not seem to work. As you can see.

1:52:05

The reason behind that is we added the package  of razor runtime compilation. Even though these

1:52:12

are razor pages, if you examine the program class  file, we are only adding controllers with views,

1:52:19

we will have to make one small change here on  builder dot services, we will have to add razor

1:52:26

pages. And on that we have ADD razor runtime  compilation. I don't know why it is like that.

1:52:35

I believe in future, they will also add this with  the views. But right now this is only available

1:52:40

with razor pages. So this is one extra  line that we have to add in an MVC project

1:52:47

to get that runtime compilation. With that  change. Let's run the project one more time.

1:52:56

And great, we see the new navigation menu right  here. We don't want everything else here. Let

1:53:02

me comment that, we'll just add the whole and  category. Let me go back here to underscore

1:53:08

layout. And I can just copy the whole. Let  me cut this, I can paste that right here.

1:53:23

Scroll down by we have the other category  controller, cut that and paste that

1:53:31

instead of features. Perfect, I will  remove these two, the one end drop down,

1:53:38

I can just leave that in comments. So that  in future if we want to drop down, we can

1:53:43

use that. And I can remove the nav bar altogether.  Let's go back and refresh. And great. This time,

1:53:51

we did not have to rebuild the application. The  text here is not visible. So let me go back.

1:53:59

And we don't want the text dark  anymore. We can remove that.

1:54:04

And perfect looks much better. If you click on  homepage, it loads the homepage and category

1:54:11

should display all the categories. Great. I  also want to hide this border. So that should be

1:54:18

inside footer. We have border top, we can remove  that. And we can give it a background of BG dark.

1:54:28

Let's go back and refresh. And this looks much  better. So with this now we are using Bootstrap

1:54:35

five in our project and we are using the boots  watch theme. Let's continue in the next video.

Bootstrap Icons

1:54:45

Inside our index view, what we can do is we can  add a button here to create a category. So if

1:54:52

we go back to what index, let me add a little bit  of more designing with Bootstrap. But we'll add a

1:54:58

div give it a class Have container and padding  of three. These are just the bootstrap classes

1:55:06

in there, we will add a div, give it a class  of row and padding top of four that may add

1:55:13

attached here, save that and go back. That will  create a new row right here, I want to divide

1:55:20

this row into two parts. By default bootstrap  divides each row into 12 parts if we want. But

1:55:28

if we just want to divide it into parts, then  we will have six in one and six in the other.

1:55:35

So here, we will add a div, give it a class  of column six, that will combine the first six

1:55:43

in one div. And we will add another  div, give it a class of column six.

1:55:50

So that will divide the screen into parts. The  first one we want to display and heading. So

1:55:56

h2, give it a class of text primary. And  we want to display category next year.

1:56:07

Let me save this, go back and refresh. And  looks good. Let me close this container

1:56:14

at the very end. So we'll close that div at the  end. And right here, we want to add a button

1:56:22

right here, we want to add a link inside then  we want to go to a new action method that we

1:56:28

will create inside the category controller  that will be for creating a new category.

1:56:35

So right here, we will say that  should go to the same controller,   which is the category controller. And it should  go to an action method, which will be create.

1:56:47

We do not have that yet. But we will create that  in the next video. We can also add couple of

1:56:54

bootstrap classes btn btn primary. And here we  can say create new category. Let me save this,

1:57:05

go back and refresh. So we have our button.  Let's align this button on the right hand side.

1:57:11

In bootstrap five, we have text and we will save  that and that will bring it to the right side.

1:57:21

Then let me just leave couple of lines here, I can  just add few br tags, then this looks much better.

1:57:30

Now I also want to use some icons here  like font awesome. But with Bootstrap five,

1:57:37

they have given their own sets of icons. So why  don't we use that? If you go to get bootstrap.com,

1:57:44

we have the icons tab. Getting started with all  of these icons is super simple. I used like font

1:57:51

awesome, but you had to register and do lots of  thing. This is available quickly, you just click

1:57:58

the install button and you have the CDN you copy  that go to the underscore layout styles will paste

1:58:08

the CDN. With that the icons are readily available  to us. Let's go back to the Bootstrap. And right

1:58:17

here, let's search for plus, if you scroll down,  we have this plus circle. If you click there, we

1:58:25

have the link here with a copy. And we just paste  it before create category. Let me add a space.

1:58:33

Let's save that. And you will see the icons are  working perfectly. Super easy to get started. With

1:58:41

that we have added icons. But if you click the  Create New category, you will see page not found

1:58:49

that is because it is going inside category  controller Create Action method. But inside

1:58:55

category, there is no action method with the name  of create. So let's add that in the next video.

Create Category View

1:59:04

That may copy the index action method.  And we can paste that one more time,

1:59:09

we will have to change the name here to create  and this will be a get action method. Now when

1:59:16

someone hits the create button, we will give them  the option to enter their name and display order

1:59:22

and create a category. So when the view is  being loaded, you do not have to pass any

1:59:28

model. You can keep that blank and then you  can create a model directly inside the view.

1:59:35

What do I mean by that? Let me create a view  here. We will right click Add View razor view.

1:59:43

We will keep that empty and looks good. Let me  add that. Now previously when we were working with

1:59:50

index, I said that this is the model that will be  passed from the controller. That is true, but it

1:59:57

is not always the case. In our create, we have not  passing anything from our controller. But we still

2:00:05

want to work on the category model. And we want  to fetch its properties when we submit the form.

2:00:12

So for that, also, you will have a model for  this view, that model will be the category model.

2:00:20

So the model that you write inside view is  not always the model that is passed from the

2:00:25

controller. But if you are passing a model from  the controller, it must match what you have inside

2:00:31

the model in the view. But if you are not passing  anything inside the controller, like and create,

2:00:38

then you can find your view with a model based  on the data that you're collecting on the page.

2:00:44

Here we are collecting the name and display  order, we will use category so that we can use tag

2:00:51

helpers to find everything for us. Now, you might  be thinking, what is this new term tag helpers,

2:00:58

I will walk you through that. But before that,  let me create a form here, give it a method

2:01:04

of post, because we will be posting our  data, because we want to create our category.

2:01:12

In that I will create a div, give it a class  of border padding of three and margin top

2:01:18

of four. In there, I will add a div, give it a  class of row and padding bottom of two. And we

2:01:28

will add a heading here, which will have a class  of text primary for the yellow color. And we will

2:01:35

display the heading which is create category. Let  me add an HR here, save this and run the project.

2:01:44

I want to see the view while we are building  that. So if we go to our category now. And if

2:01:51

we hit the Create Category button, it will take  us to the Create category view that we created.

2:01:59

Here, I want to display a label and a text box.  So outside of this tab, I will add another div

2:02:07

give it a class of margin bottom three  in there, we want to display a label,

2:02:13

you can just do label and give it a name like  this. That will work. But what we want to do

2:02:20

is we want to bind everything on this page  with our category model. Because of that,

2:02:26

the dotnet core team has provided tag helpers,  which starts with ASP. So here we have ASP

2:02:34

hyphen, four. When we add that in a label, we  can use any of the properties from our model.

2:02:41

So you can see it is already displaying created  date, time, display, order, name, and ID,

2:02:48

we will select name here, and we do not have to  populate anything else. If we go back and if we

2:02:55

refresh, the output will be seen. Because what it  does here is it displays the name of the property

2:03:03

inside our category model. So we have it as  name. So that is what it will display in the UI

2:03:10

as well. Let's go back to the CREATE VIEW. And now  we want an input field. So we will say input. And

2:03:19

we will just say ASP for Tag Helper, this input is  for name. That way it will do all the binding. And

2:03:27

when this form is posted, it will post an object  of this category class with the name populated,

2:03:35

we can give it a bootstrap class of form control.

2:03:41

Let me save that and refresh. Whoops  I close that. Let me run it again.

2:03:50

And create you can see a text box right here.   This looks good. Let me also add a text box  for the other field which is display order.

2:04:01

So I will just copy this, paste it here as before  will be display order this time. Let me save that,

2:04:11

go back and refresh and create this looks good.  You can notice the name of the label is display

2:04:18

order without a space. And that is because inside  category that is the property name itself. We will

2:04:25

fix that in just a second. But I wanted to show  that now let me add a button here to actually

2:04:33

submit the form and I will add a link to go back  to the Index page of the category controller.

2:04:40

So we'll go to create here and we will  add a button give it a type of submit.

2:04:48

We will also give it some bootstrap classes of btn  btn primary and that will give it a width of 150

2:04:56

pixels. I will call this button as Create.  Next I want to add a link here. This link,

2:05:04

I want to go back to the Index action method  inside the category controller to load all of

2:05:10

the categories. So it is more like a back button  link. So first what is the controller name,

2:05:17

that is the category controller. And what is the  action in there, that is the index action, we will

2:05:24

give it some bootstrap classes btn btn secondary,  and I will also give it a style with the width

2:05:32

of 150 pixels. Within here, we will display  back to list. Let's go back and looks better.

2:05:43

If we click on Back to list you can see that is  functional. Now, in the next video, we want to

2:05:49

hit the create button and create our category.  With that let's continue from the next video.

Demo- Create Category

2:05:58

No, I want to create the category when we populate  the details and hit the Create button. So when we

2:06:06

hit the create button, we will have to create a  post action method inside the category controller.

2:06:13

And in that post action method, we will already  be fetching the category object that is populated

2:06:20

because we used tag helpers. So inside the  category controller, we can copy this create

2:06:27

and paste that one more time. Let me stop the  application. Inside the parameters we will be

2:06:33

receiving a category object, let me call that Obj.  This will be post action method. Now if an action

2:06:42

method is post, we have to write the attribute  HTTP POST. On top of that the dotnet team

2:06:49

has approved and then we also have to validate  anti forgery token anti forgery token is there

2:06:58

to help and prevent the cross site request forgery  attack. What it basically does is inside any forms

2:07:05

that you have inside the application, it will  automatically inject a key there. And that key

2:07:11

will be validated at the step. That key must  be valid to prevent cross site request forgery.

2:07:19

I have explained the cross site request forgery  and the Validate anti forgery token in much

2:07:25

details inside free content on dotnet mastery.  If you search and look for cross site request

2:07:32

forgery, you can find that video on YouTube and  watch that I do not want to change the focus of

2:07:39

this course. So they will just validate  that token on all of the post request.

2:07:45

This is not required. But I will highly recommend  that to avoid the cross site request forgery.

2:07:52

Now once we have the category object that is  populated with name and the display order,

2:07:58

we want to create that record inside the database.  So to do that, we will use our DB context.

2:08:05

And on there we have the categories.  Then in order to retrieve categories,   we did not have to write anything else. But  when we have to add something to the table,

2:08:16

we have the add method there. And you can  see it expects a category entity. We already

2:08:22

have that inside the OBJ. That is what the user  populated. So we will add that to our database.

2:08:30

Now once you add it to the database, it is not  pushed to the database right now. It will be

2:08:36

pushed to the database. When you run the command  underscore DB dot Save Changes. At that point it

2:08:43

goes to the database and saves all the changes.  Once the changes are saved, we have returned view

2:08:50

here. That will take us back to the same category  view. Let's see when we are done. Let's go back

2:08:57

to the index so that we can see the new category  that was created. So rather than return view, we

2:09:04

want to redirect to an action, we want to redirect  to the index action, it will look for this index

2:09:13

inside the same controller. If you had to go  to an action method in some other controller,

2:09:19

you can define the controller name right  here. But since we are in the same controller,

2:09:25

we can just mention index and that will work.  Let's run our application and try this out.

2:09:33

Let's go to our category. And let's create a new  category. I will call this Test display order 12.

2:09:43

And create you can see it has been created. If  we go back to our SQL Server, let me close this.

2:09:51

And if we do select top 1000 Now we see  two records. So perfect. This is working as

2:09:58

expected. We are able to create a new category  now. But our create category is not perfect.

2:10:06

If you hit the create button, you will run  into an exception. Let me switch back to the

2:10:12

project. And you can see the exception is  cannot insert. Now, inside the column name,

2:10:18

we did not have any validations. But inside  database, we said that this was a not nullable

2:10:24

column. So when the Entity framework code tries to  save, it gives the exception, pretty smart. But we

2:10:32

have to be much smarter to handle the validations.  Let's take a look at them in the next video.

Server Side Validation

2:10:41

Now when we try to create a category without  any fields populated, right now we see some

2:10:47

error message because it cannot add an object  when in the database, we have validation.

2:10:53

But that is not a good approach. Because we are  throwing exception. In that case, what should

2:10:59

happen is we should handle our validations  inside the server side, which is controller

2:11:06

as well as inside client side, which is inside  the CREATE VIEW. So how do we handle validations?

2:11:13

Let's first do that at the server side. So when  we receive a model here, we can check whether this

2:11:20

model is valid or not. And what defines a valid  model is the validations that we have right here.

2:11:28

name should be a required property. So let's go  back. And let's make sure that our model is valid.

2:11:35

In order to check that with dotnet core, we have  model state.is valid, that will determine if the

2:11:43

model is valid or not. If that is valid, we want  to create and redirect to index. If that is not

2:11:50

valid, we just return back to the view with the  object. Let me run this and show you what happens.

2:11:58

I'll add a debugging point on if the model  state is valid. So let's run our application.

2:12:06

Let's try to create a category with no fields  populated. We hit our breakpoint. And if you

2:12:13

examine the is valid flag is false. So  our model state is not valid. Great.

2:12:19

But now which property in our model state is not  valid? It is important to know that because right

2:12:25

now we just have two properties, it could be  possible we have over 50 properties in a model.

2:12:31

So in that case, you hover on model state, you  expand that inside the error counts, you see we

2:12:38

have two errors. If you hover on the values here,  Result view, you will see which properties are not

2:12:46

valid. You can see display order is not valid, as  well as name is not valid. Let's continue here.

2:12:54

And this time let me populate display order  and hit the create button one more time,

2:13:00

model state will still be not valid. But  this time if you examine the error count

2:13:05

is only one because we already fixed one of  the errors by providing display order. So now

2:13:12

you can see display order is valid, but name is  still invalid because it is a required field.

2:13:19

That's it server side validations are working.  But with that it would be helpful if we are

2:13:24

displaying an error message right here,  so that user can know why it is failing.

2:13:31

Doing that is super simple. We will remove the  breakpoint here. And let me go to the View.

2:13:38

Just like we have ASP for Tag Helper, we  also have a Tag Helper for validation.

2:13:45

So inside name, we can add a span tag. And we  have the Tag Helper ASP validation for on this

2:13:53

you have to define the model. So we want to check  validations for name here. And if there are any

2:13:59

errors, we want to display them in red color. So  we can give it a bootstrap class of text danger,

2:14:06

we will copy this span and we will paste that  for display order as well. With that in place,

2:14:13

make sure to save this and we will go back and  refresh the page. Let me just reload this. And

2:14:21

this time if you try to hit the create button, you  can see the default error message is displayed.

2:14:27

The first one that we have is name is a required  field and display order value now is invalid

2:14:34

because it's an integer. So with that we can  display the default error messages right here.

2:14:41

So with that we have our server side validations  in which it checks at the controller level if the

2:14:47

model state is valid or not. If it is not valid,  it populates the model state with error message.

2:14:54

It returns back the obj and it automatically  displays the validations that we have added,

2:15:01

we did not have to write JavaScript or any  other complex code. It is doing all of that,

2:15:06

with the help of tag helpers. Let's continue in  the next video. With server side validations.

Custom Validation

2:15:15

What if you want to display a summary at the top with all the error message, if you want to do that

2:15:22

dotnet core has a solution for that. You will need a div this time and not a span that you were using

2:15:29

with validation for. And inside tip, you can see we have a Tag Helper ASP Validation Summary,

2:15:37

this will be all in this case, it will display all the validations at the top in a summary format.

2:15:45

Once we make this change, let's go back and refresh here. If you hit Create rate, now you see

2:15:52

the error messages at top as well. When you select all it will display models as well as properties.

2:16:00

If you want to limit that you can change that to model only and none. But if you are

2:16:05

using Validation Summary, I personally like all, and this is great. Now along with that,

2:16:11

what if we want to add some custom validation? Like let's say inside the controller here, we want

2:16:17

to make sure that we do not add any category which has the same name and display order.

2:16:24

How can we add that to the validation. So before we confirm if the model state is valid,

2:16:30

we can check here if obj dot name is equal equal to OBJ dot display order, we will have to convert

2:16:38

that to a string. If this is the same, we want to tell our model state that this is not valid.

2:16:45

So for that on the model state itself, we can add custom model errors. Here we have

2:16:51

a key and value pair, you can give any key name that you want, we will call this custom error.

2:16:58

If you want you can also use string dot empty. But since this is a key, you have to make sure

2:17:04

that you do not use the same key two times. Let me add a custom error that the display

2:17:11

order cannot exactly match the name. Now since we are making the change inside the controller,

2:17:17

we will actually have to restart our application to see this change. If we were working with the

2:17:24

view, we did not have to restart. Let's go to our category and try to create a category with the

2:17:32

same name and display order. This time, you can see we have our custom validation that is being

2:17:38

displayed in the summary. Now if we did not have summary, then this would not have been displayed.

2:17:45

But what if you want to display the error message inside name as well, it is pretty simple, we will stop the application and the key, we will just change that to name.

2:17:57

Because inside the category object, if you notice, we have name. So inside controller,

2:18:03

we are adding a new error inside the name property. Let's run this and give that a try.

2:18:11

Let's go to our category and try to create a new category with the same name and display order.

2:18:19

Great. Now you see it displays input cases. So with this, we have seen how to add custom

2:18:26

error message. But all of the validations that we have so far are done on the server side.

2:18:33

Every time you hit the create button, you will see the page will reload. So that means we are hitting server every time. Even if the name and display order are empty.

2:18:45

We go back and hit the server because the page reloads. What if we want to make all of this validations on client side? Let's do that in the next video.

Add Client Side Validation

2:18:58

Now if you want to do the basic model validations like name and value for the display order on the client side, it is pretty simple. If you examine inside the shared folder,

2:19:09

we have a validation scripts partial that is using jQuery dot validate. It is inside a partial view.

2:19:18

So if we want to do client side validations we just have to include this partial view

2:19:24

in our view. So that brings the big question. How do we use a partial view, we have not created one,

2:19:32

but we are using the one that is already available. It's not a rule, but typically it's

2:19:37

good to name partial views with underscore. So even if in future when we create a partial view,

2:19:44

we will follow that convention. Now in order to include a partial view it is pretty simple.

2:19:50

You can directly say partial and then you have to write name of the view. So make sure to type the

2:19:57

exact name if you have a special mistake there, then this won't be included. With that the errors

2:20:04

go away. But inside the partial view, if you open up, we have script files. So we need to add that

2:20:12

inside a script section of our view. So here we will have to create a section for scripts.

2:20:20

And in there, we will have to add our partial view, and that to inside razor syntax.

2:20:27

Once you add this, let's run the application one more time and see the magic. Let's go to

2:20:34

our category. And let's try to create a new category, we will hit the create button,

2:20:40

you see, we get the validations. But we are not going to server this time. All the validations

2:20:46

are done on client side, because of the script that we included. To double check, you can just

2:20:53

add a debugging point here and hit the Create button. It will never reach our debugging point,

2:21:00

it will reach our debugging point if the name and display order are the same, then the client side validations are valid. The validation that fails is the server side custom

2:21:11

validation that we added. And if you continue, perfect, you can see that in action. So with this,

2:21:19

we can see how easy it was to add client side validation. Let's continue in the next video

Display Name and Range Validation

2:21:29

that we have added the client side validations I want to show you one more thing. Right now we have

2:21:35

the default message that is being displayed. Right now we have the default message that is

2:21:40

being displayed. On top of that you can see right now in display order, there is no space that is

2:21:47

available. And it is the same with the validation. The reason behind this is if we go to our category

2:21:54

model, you can see that's the property name. And that is exactly what is being displayed. So if we

2:22:01

want to display something that is not the same as property name, we have display name here,

2:22:07

which is inside component model. And here we can just give whatever name we want to display. So

2:22:14

let's add a space there. And it will correct that automatically. Let's run this and give that a try.

2:22:25

Let's go to our category, create category. And this time you see in errors, we have a space

2:22:33

labels, we have a space, and perfect. Looks good. Now if you go on the official documentation,

2:22:41

there are quite a few data annotations that you can use. We see the required one that we saw,

2:22:47

we also have a range attribute. I won't go into all of them, you can explore them when you want,

2:22:54

you can see the display attribute that we used. Now let's just try one more, which is the range attribute that specifies a range for a field. So let me just close this,

2:23:06

and we'll stop the application. Inside our display order, we will add a range attribute.

2:23:12

And here we have minimum and maximum value. So let's say minimum is one maximum is 100.

2:23:20

So now if the display order is not in the range, it will give an error message. But

2:23:26

the error messages that we see on the screen with validations are also customizable. So with range,

2:23:33

if we want some other error message, we have that property right here. And we can give

2:23:39

a custom error message that will have our custom error message as display order must be between

2:23:46

one and 100 only with two exclamation. Let's run our application and see that in action.

2:23:58

Let's hit Create we have our required one, let me add something more than 100 and you can see it

2:24:05

directly displays our custom validation. With that you can see everything is working as expected.

2:24:12

And we saw how to add custom error messages, range validation and how to use display name.

2:24:20

Now inside our CREATE VIEW, I will hide or comment the Validation Summary

2:24:25

because we don't want that right now. The inline validations are sufficient.

2:24:31

Let me go back and remove the debugging point. Everything looks good so far. From the next video

2:24:38

we want to work on editing and deleting our category. Now that we have the Create

Edit Category - GET

2:24:45

functionality working as expected, let's work on the added functionality.

2:24:51

For added we will go to category controller and just like we have the Create get and create post.

2:24:58

We will copy that And paste it one more time. This time, we'll we'll call them for any.

2:25:06

So let me change the action method names here. One thing that will be different for edit is when

2:25:12

the page is loaded, it will display the existing functionality of the category that was selected.

2:25:20

Here, we will retrieve an integer, which will be ID. Based on that ID, we have to retrieve

2:25:26

the category details and display them. So we can check here if Id is now or if ID is equal to zero.

2:25:37

In those cases, we will return back not found, because that is an invalid ID. If that is not the

2:25:44

case, we will retrieve the category from database or category from dB. And we will extract that

2:25:51

using our underscore DB dot categories. Now this will retrieve complete list of all the tables.

2:25:59

So on that we have few more ways of finding one of the entity. First is single or default, which

2:26:06

returns only one element every time, it will throw an exception if there is more than one element,

2:26:13

id is a primary key, so there is no chance that there will be more than one element. Similarly,

2:26:20

we have single, the difference between single and single or default is single will throw an

2:26:26

exception if no elements are found for that ID, but single or default will just return empty if

2:26:33

there are no elements. So default will not throw an exception if the element is not found. Similar

2:26:40

to single or default, we also have first and first our default, the difference between single

2:26:47

or default. And first, our default is that single or default will throw an exception if there are

2:26:53

more than one elements. First 34 will not throw an exception, it will return the first element

2:26:59

of the list. So if more than one elements are found, first, our default will return one element

2:27:06

singular default will throw an exception. Last but not the least, we have one method which is Find.

2:27:13

Find basically tries to find that based on the primary key of the table, we know that ID is the

2:27:19

primary key. So we can use Find here and pass the ID. It we'll find the category based on

2:27:26

that ID and assign that to our variable. Let me also show you how we can use first our default,

2:27:34

with find you just had to pass the ID with first our default, you have to pass an expression here

2:27:40

where we will have a generic object, let's call that u. And we will say u goes to u.id

2:27:47

should be equal to the ID that we have, if that matches in any of the categories will

2:27:53

retrieve that and give me the first one out of them. Let me copy this and paste it one more time.

2:28:00

For single or default, the syntax will be same, we will just use single our default. And let me

2:28:07

call this variable single. So these are three ways to retrieve a category from database based

2:28:14

on the ID. I will comment two of them out, but I wanted to show you how that could be done using

2:28:22

Entity Framework core. Now once we retrieve the object, we will check here if category from dB.

2:28:29

If that is now that means again, we will return not found. If we find that category,

2:28:36

we will return that category to the view. So next what we will do is we will create this view

2:28:43

which will have the category loaded. And we just need to display that that view will look exactly

2:28:50

same as the CREATE VIEW. But the only thing different will be that it will be for edit page.

2:28:57

So we can copy the view that we have for create insert category folder and paste

2:29:02

that one more time. I can rename this to edit to give us a head start. Let's open that edit view

2:29:11

our model will be the category model, but rather than create category, this will be added category.

2:29:18

Now when we added a category, we want the validations. So that will stay the same. The

2:29:25

button text we will change that from create to update. And it is just submitting back.

2:29:32

We do not have which action method it should submit to. If you do not provide any name here,

2:29:38

it will by default submit to the same action that is of the name of the cat. So since the

2:29:44

get is from edit, it will post back to the same action method which is added.

2:29:50

But if you want to be explicit about the action, you can use that with the Tag Helper ASP action.

2:29:57

And you can write edit like that. So if you want you can be explicit, but that is not required,

2:30:05

the backdoor list will stay the same. So with that in place, if we want to see this in action,

2:30:12

let's go to the index view of category where we are displaying everything, I want to add a link to

2:30:18

navigate to the Edit action method. That will be three columns here, let me add another th tag. And

2:30:26

inside the TD, we will have one with the links. So inside this TD, we can add a div with a class

2:30:34

of ID, 75 and btn group, we can also give that a roll of group in there, let me add an anchor tag.

2:30:43

And we will use tag helpers inside this anchor tag. This will be for edit. And if we go back,

2:30:50

and if we open up the bootstrap icon, let's try to find an icon for edit, we have this pencil square,

2:30:58

let me copy that and paste that here. Let's run the application. And we will get back to the Index

2:31:06

in just a second. While the application loads, let's get back here. So this added should take

2:31:13

us to category controller and edit action method. So if we go back here, we want the ASB controller,

2:31:22

it has to go to the category controller, ASB action, it should go to the Edit action.

2:31:31

That is great. But if we go take a look at our controller action method, it also expects an ID.

2:31:39

So we have to pass that ID with our link. We can pass custom wave tables here. So we can say ESP

2:31:47

route. And then what is the name of the variable that is ID right here. So we will just say ID

2:31:55

here, this will have to bind with our object. So we'll say add, we have obj.id. You can see

2:32:05

how dynamic we can get with the razor pages. With that. Let's go back and refresh. Go to category,

2:32:13

we should see the edit button. Yep, that is here. You can see if you hover on edit, you

2:32:19

can see the URL where the ID is changing. If you click edit here, it will take us to edit category,

2:32:27

and it automatically populates the name and display order. Our back to list is also working

2:32:34

as expected. To make this button look pretty, we can just add two CSS classes btn btn, primary

2:32:43

and some margin. Let's go back and refresh. And this looks much better. So I added cat action

2:32:51

method is working as expected. In the next video, let's work on updating our category.

Edit Category - POST

2:33:00

Now we are able to load the category that we want to edit. When the user hits the Update button,

2:33:06

it will go to category controller, we have the post action method for edit,

2:33:12

you can have the same validation that we had there. But if all the validations are valid,

2:33:18

we do not want to call the add, we want to call the update method on our categories DB

2:33:24

set. This is a built in method with EF core where based on the primary key, it will automatically

2:33:31

update all of the properties. And you do not have to do any manual update, it will take a

2:33:38

look at the obj right here and it will find its primary key retrieve that from the database,

2:33:45

it will check all the other properties where the values have changed, and it will update all

2:33:50

of those. So you can see it is handling all the complexity. And we just need an update statement

2:33:57

to update any of the category based on the ID. After that, make sure that you are saving the

2:34:04

changes and we are redirecting back to the Index action. If you examine here, you can see all the

2:34:11

options that are available with RTP sent we have add remove add a thing at range that will add

2:34:19

multiple objects at the same time. We have update, and much more. For this time, we'll select update,

2:34:27

and we will have to restart the application since we made some changes to the controller.

2:34:34

Let's go to our category. And let's try to edit any of the category.

2:34:42

And great you can see update is working successfully. Now the last part that I want you

2:34:48

guys to implement yourself is to add a button or a link here to delete any of the category. Once you

2:34:55

do that, it should take you to a details page like edit category But this will say delete category

2:35:02

and the fields will be disabled, then that will be a button to delete. And when you click that

2:35:08

on the post, it will delete the category and take you back to the Index page. It is exactly same as

2:35:16

what we did with edit category. But rather than update, you will have to find what keyword to use

2:35:22

to delete the category. Also, I want to show you that inside edit our validations are working. If

2:35:29

you hit update here, you can see it does not let you update your callback, the data stays

2:35:35

the same. So all the validations are working as expected. But in the Delete category,

2:35:41

we will not need any validations because you have to make sure that the fields are disabled.

2:35:48

So good luck with the assignment. And I'll show you how to do that in the next video.

Delete Category

2:35:55

Last functionality we want to work on is the Delete functionality inside category. So we

2:36:02

will copy edit as as, and we will paste that one more time both GET and POST,

2:36:09

I will change this to delete. And we will use the same name right here for the post action method.

2:36:16

When we are deleting a category, we want to retrieve that category. And we want to display

2:36:22

that on the view. So this functionality stays the same with the get action method. Now with

2:36:28

the post action method, you can either push the complete object, or you can just pass the ID here.

2:36:35

So we will have integer ID. And based on that ID we will first retrieve that category. Now I do not

2:36:43

need any model state validations. So I will remove them, all I have to do is based on the ID, I have

2:36:50

to retrieve the category. So I'll say variable obj is equal to underscore DB dot category start find

2:36:59

based on the ID we will find that if there is no we will return not found. Else on categories

2:37:06

we have removed. And we will remove the object, save the changes, redirect back to the Index page.

2:37:13

That looks perfect. But we see an error with the name, we cannot have same signature with the name

2:37:21

and the parameters for to action method. So far. Now let me rename this to delete post.

2:37:29

Let's work on adding the Delete view it will be same as added because we will be displaying them. So let me copy edit, paste it one more time, I will rename this as delete.

2:37:41

Let me close all the other views so that we don't confuse ourselves. This will be delete category

2:37:48

and the button let me make it danger. Since we are deleting, we do not want any validations and

2:37:55

the fields will be disabled. So let me copy that as well. Now the post action limb is added. But

2:38:04

we have renamed the POST method to delete post. So we will copy that and paste that right here.

2:38:12

Let's run our application and give this a try. Let me go back and add a debugging point right here.

2:38:22

Let's go to our category. And we forgot to add the delete button

2:38:27

inside the index view. I will copy this and paste it one more time. This will be for delete,

2:38:35

rather than square we will have to look for delete icon. So let me look for a trashcan.

2:38:45

The get action will be delete. So we will use that. Let's go back and refresh. We have edit and

2:38:52

delete. That may give it a different color. And perfect. Now if you try to hit the delete button,

2:39:01

it will take you to the Delete category. Let's see what happens when we hit Delete button.

2:39:07

It will hit our breakpoint and you examine the ID is now. Now why is this ID No,

2:39:14

it was not now when we were editing button delete this is no. The reason behind that is

2:39:21

inside Delete. Let me hit continue here. And it will take us to not found page that's perfect.

2:39:28

But the reason behind that is inside Delete. All of the fields for our model is disabled. If one

2:39:34

of them was enabled, it is primary key. That's why it was automatically being set on the forum.

2:39:42

So in this case, what we will do is we will have an input field with the Tag Helper ASP for ID.

2:39:48

That's the only field we care about. And we will keep it hidden since it is inside the form when

2:39:55

the form is posted. We will retrieve that ID right here. Let's go back. And let me refresh.

2:40:04

Let's hit the delete button. This time the ID is populated. So now if you hit continue,

2:40:11

now, if you are editing, you make some change, you get this error message. But I believe with Visual

2:40:17

Studio 2022, they are just giving that right now. And it will be fixed later on. In that case,

2:40:24

just rerun the application. So let's go to category here and try delete one more time.

2:40:32

Perfect that category was deleted. But that delete is working as expected. But I want to show you

2:40:40

one last thing. Right now inside delete, we have the action as delete post. What if we want it to

2:40:47

be Delete? In this case, it won't work because the controller will know that the action method

2:40:53

name here is delete post for post. So if you want to explicitly give that a different name,

2:41:00

we have action name attribute right here. And we can give it delete. That way the controller will

2:41:08

know that if a request comes in for delete action method HTTP POST inside the category controller,

2:41:15

that is this particular action method. So let me run that and show you that this also works.

2:41:23

So let's go to category and delete one more category. Perfect.

2:41:31

So with that all the functionality are working as expected.

Tempdata

2:41:37

What to display some alerts when someone deletes edits, or create a category on the category

2:41:43

index view? How will we do that? For that we have something called as temp data in dotnet core

2:41:51

temp data has been provided with one single purpose, whatever we want to store in temp

2:41:56

data stays there for only one request. After that, if you refresh the same page,

2:42:03

that will be gone. So that is perfect for displaying alerts of successful or failure

2:42:09

qualification. Let's take a look at how we'll be doing that. Let's say right now that when we

2:42:15

create or edit or delete anything, we will add something to attempt data of success.

2:42:21

So once the Create is successful here, before we return back, we want to use temp data,

2:42:28

it is directly available as you can see, so temp data and then you have to give any key

2:42:35

inside there. So let me call this temp data of success. And we will store a message there

2:42:42

that category created successfully. On create post. If they are editing admin store category

2:42:50

updated successfully. In delete, I will say category deleted successfully. So we are

2:42:58

storing some string inside temp data with the key of success. Now where do we want to extract this,

2:43:06

we want to extract that inside the index view. So at the top here, we can check if temp data of

2:43:13

success is not no. The main important thing is the key name must exact match. If you use a different

2:43:21

key name, this will not work. So if this does match inside an h2 tag, let me display temp data.

2:43:31

And the value inside that. Let's save this and run our application to see what happens.

2:43:40

With that change, let me try to create a new category.

2:43:47

Great, you can see we have category created successfully. Now since this is in temp data,

2:43:53

if you refresh, this will go away. So it only stays in memory for just one redirect. And

2:44:01

after that it goes away. If we try to edit any of the category, we get the error of course,

2:44:09

we see category updated successfully. When we delete, we get the deleted successfully

2:44:16

if you refresh that goes away. So this is great when you have to display notifications on some of

2:44:23

the actions that are performed. Now this temp data can be used throughout the application

2:44:30

and not just one page. So why not when we are checking this temp data success and displaying

2:44:36

that do that at a global level. So we do not have to add this in all the pages for

2:44:43

that partial views will be a perfect candidate. We can have some code inside the partial view

2:44:49

and we can call that partial view everywhere we fought because right now it is four lines of code,

2:44:56

but it is possible that this code will increase drastically. When you do something fancy

2:45:01

for notification. So let's move this code inside a partial view and see how we can count that.

Partial views

2:45:11

Now that we have added a code to display some of the alerts and notification, we want to make sure that this code is applicable on almost all the pages. Because if in future,

2:45:22

we add more pages, we don't want to copy and paste the same code in other places.

2:45:28

So the best thing is inside the shared folder, we will create the partial views.

2:45:33

That is the folder that I like to place all of my partial views, we will start the application,

2:45:40

right click Add View, we will go with the razor view, not the empty one. And here,

2:45:47

we will select Create as a partial view, we did not do that earlier. But now we will do that.

2:45:55

That will create an empty partial view. I forgot to give that a name. So I'll have to rename this,

2:46:02

once it is created. It's just a view, we will change that to underscore notification.

2:46:10

What we want to do is the temp data check that we have here. Let me copy and paste that

2:46:16

here. And it will do its check. Since we have a temporary data for success, we can also implement

2:46:23

a temporary data for error. So we can display both success and error notifications. So great.

2:46:31

Now inside index, we can just say partial here. And we can say name is equal to underscore

2:46:40

notification. Pretty clean, right? Let's say in future, you want to add this to multiple pages,

2:46:47

you just have to add this partial tag. And that will be done. Let's run our application and see if our partial view is working as expected.

2:47:02

That's quiet to create a new category. And grid that is working. If we refresh

2:47:10

that goes away. So with that our partial views are working as expected. Then displaying the

Toastr Alerts

2:47:20

text as an alert. Why don't we use something fancy like toaster. So we have toaster js.

2:47:29

So with toaster difficult that GitHub page, we have a JavaScript notification that we can use.

2:47:36

If you go to demo here and hit show toast, this notifications are much cleaner and easier to read.

2:47:44

In order to use that, we will copy the minify js and css files. Let's add that to our project.

2:47:52

In the underscore layout of the CSS file, we will be adding right here.

2:47:58

So we'll say link, Arielle is equal to stylesheet. And the href, let's go back and copy just the CSS.

2:48:10

So we will copy this and paste it right here. Now the GS file will be required

2:48:17

at the underscored notification level. So we will have to go back, copy the GS file.

2:48:24

And we need to include that right here. So we will have to use a script here, SRC.

2:48:33

Let's paste that. Also, when we are using a toaster, we will need the jQuery reference

2:48:39

that is inside an IB we have the jQuery di st jQuery dot main dot j s. So we will have to add

2:48:48

both of them in both the places. Next we need the toaster alert. So if we go back,

2:48:56

we should have a usage right here. We have toaster dot warning toaster dot success, we will use it

2:49:03

exactly like this. So if we go back here, if we have success that is populated, we need a

2:49:11

JavaScript code. So we'll add a script tag. We will say type is equal to text forward slash

2:49:19

JavaScript. And I've used toaster dot success and display everything inside temp data of success.

2:49:27

Make sure here you are using the single quotes. And then you need to activate sign for accessing

2:49:33

the C sharp variable. Let's copy this clip tag and paste that for error. And this looks great.

2:49:44

So with that we have added toaster Jas in our website. If we go back even while the application

2:49:51

is running, these are just the JavaScript changes. So you can refresh the page here

2:49:58

and let's try to edit some thing in the category. And great, we see our toaster alerts here,

2:50:05

and everything is working as expected. Now that say this notification is something you will have

2:50:12

across all the pages. In that case, it does not make sense to add this one line in all the pages,

2:50:20

even though it's just one line, it just doesn't make sense. So rather than adding this in all the

2:50:26

pages, you can add that in underscored layout. And that will be included in all the pages.

2:50:34

So just before render body where we have all the body that we render for the page, we will paste

2:50:40

our partial view. And that will make sure that it is included in all the pages of your application.

2:50:47

So make sure that you include only things that are needed across all the pages, because this will be loaded every time your page loads. So with that, if we go back and

2:50:59

refresh and try to delete a category, our toaster alerts are working as expected.

Scaffold CRUD

2:51:08

Now that our application is up and running, and everything looks scared, I want to show you one

2:51:14

thing with the controller, we added the controller ourselves and we also created all the views.

2:51:21

But with dotnet core, we have some help if you want. So if we try to add a controller, this time,

2:51:28

we will implement the same logic work with MVC controller with views using Entity Framework.

2:51:36

We will hit add here. And this will create an MVC controller with views using Entity Framework on a

2:51:44

model. So we will select our model category. On that we want the CRUD operations. Let's select

2:51:52

a data context that will be used to access the database. And we will give our controller a name.

2:51:59

I will leave the default name. But rather than categories, I will say categories term controller,

2:52:06

since I want to delete that later on. Let's add this and see what happens. This will

2:52:14

scaffold everything for us, it will create a new category stem controller, and it will create

2:52:20

the views for create, edit, delete details and index. You can see the new controller,

2:52:28

it has all the action methods as well as the post action methods. And inside the categories temp,

2:52:35

it has added five views. Let's run the application and see the change.

2:52:43

In order to navigate there, we will have to manually type category stem

2:52:54

it takes us to the view here where we have the index view, it displays our category, we can create a new category here.

2:53:05

With this, it also asks us to input the created date time, let's do that you can see it is not as efficient because it does not know that created date

2:53:16

time will be the default one. So it asks the user to input that. But you can create that,

2:53:22

edit that back, view the details. And you can also delete any of the request.

2:53:30

So all the crud functionalities with the views are ready for you and generated in just a minute. But

2:53:37

you'll have to spend time on customizing this based on your requirement. The reason I did not

2:53:43

want to go into this directly is I wanted you to spend some time and learn how to do everything

2:53:49

from scratch. But once you know this, you can take a look at the controller action method,

2:53:56

most of the code will be the same. But that is also an option that is available. With that,

2:54:04

that may remove the temp controller that we added as well as the views. With that our application

2:54:12

is complete. Let's see how we can deploy this to Azure and see everything live on the internet.

Create SQL Database on Azure

2:54:21

And the final task is to deploy an application to Azure. So for that, you will have to create

2:54:28

an account on portal.azure.com. I already have an account. If you are signing up for the first time,

2:54:35

you will get $200 of Free credit that you can use. The first thing that we have to do in here

2:54:43

is we need to create a SQL Server and a database inside that. So we can create our search for SQL

2:54:51

databases. And let's hit the Add button. Usually I like creating everything from Visual Studio

2:54:58

but SQL Server I'd like to create here. That way, I can easily see all the configuration,

2:55:05

you will select the subscription that you have. And in there, we can create a resource group.

2:55:12

I will call this resource group as taught at mesh tree underscore course. And then let's

2:55:18

enter a database name, we will contact dotnet mastery underscore dB, we do not have a server.

2:55:26

So let's create a server, the server name will be unique. So I will use start at mastery server.

2:55:34

And then we need an admin login. We cannot use admin here, that won't work. So I will just use

2:55:41

admin SQL for password, you can use something secure, select your location, and hit OK.

2:55:50

Now next option is to be one to use elastic pool, we will say no there and in the server

2:55:56

location that's not available. So let me select some other location HDLs too. And let's hit OK.

2:56:06

Then you need to select the compute flash storage, we will select Configure database there,

2:56:11

and I will use the lowest option which is basic here. And that is five usd per month.

2:56:18

Let's apply. I do not want to pay more for the subscription. So I'm going with the lowest option,

2:56:26

the redundancy backup, that is perfectly fine with me. Let me hit review, let's create. We're

2:56:33

going to hit Create here. And it will take a while to configure our database and set everything up.

2:56:42

And perfect our database has been created. If we click go to resource, it will take us to

2:56:48

SQL database. Now one thing that you will have to do is if you want to access this database,

2:56:55

from your local SQL Server, you will have to select the Set firewall settings and add your

2:57:01

client IP that will enable your client IP and you can access the database from SQL Server.

2:57:09

Let's hit the Save button to add your client IP there. Once that is done, let's go back to SQL

2:57:16

Server. And we need to connect to that server. So we will hit OK here. Go back to the resource.

2:57:24

And we have connection string. Let me copy the server name here.

2:57:30

Go back and paste it. And we will use SQL server authentication. We have admin SQL

2:57:37

and password. Let's press Connect. Great, we are able to connect to our SQL Server.

2:57:46

So with that in place, let's continue from the next video. That will close the other tabs here.

2:57:54

Now we are able to connect to our database from SQL Server. Let's go back to our project. And we

2:58:01

want to deploy our website to ensure for that, we will right click on the project

2:58:07

and hit publish. We will do all the setup directly from Visual Studio to create an app service.

2:58:14

And we will use the database that we created. Here we will select Azure, hit the next button,

2:58:21

we will be using Azure App Service for Windows. If you are not signed in, make sure to sign in

2:58:28

with the same account that you have used to create the database on Azure portal.

2:58:34

You will select the subscription name here, and we don't have any app service that are running.

2:58:41

So we will create a new app service here. And we need to give a meaningful name.

2:58:48

I will call this bulky book web. And because of that the final site name will be bulky book web dot Azure Websites dotnet. We will have our subscription resource group. If

2:59:01

you do not have one, you can just create one. It is just any name that you want. That is more like

2:59:07

a folder structure in which you can organize everything together. Next is the hosting plan.

2:59:14

By default, there is as one hosting plan, I want to use the free hosting plan. So we will select

2:59:21

new here. And we will select the free plan that is available with Azure. Let me hit

2:59:28

OK here and hit the Create button. This will take a few minutes, but it will create and configure

2:59:35

your app service. Once the app service is created, we will have to configure the connection string

2:59:42

and then in the new database that is created. We need to make sure that migrations are executed.

2:59:49

Perfect the system. Let's hit finish here. And inside dependencies you see we have SQL

2:59:57

Server database. You can either configured this here. Or if you just click the Edit button here,

3:00:04

it will display the database, which has a connection string, if you go back to your portal,

3:00:12

and we will have to go to our SQL database, we have the connection string right here,

3:00:19

we will copy this, and we will paste that here, we just need to make sure a few things,

3:00:26

we just need to make sure that password is populated. Let me do that here.

3:00:35

User ID looks good. We will say that we want to use this connection string at runtime. Let me copy

3:00:43

this. And inside the Entity Framework migrations, we want to apply migrations on Publish,

3:00:50

we will paste the connection string here as well. Let me go here to make sure password and user ID

3:00:58

looks good. And the last option is since we are using dotnet, six, it is not available inside the

3:01:06

API service. So for that, we will have to change the mode to self contained that we'll have a self

3:01:13

contained environment with dotnet six. With that, let me save this. And let's publish. When it will

3:01:22

try to run the migration in the logs, we will see one error with the IP address. Let's wait

3:01:28

for that. And you can see here it is failing here. So it should give us the error message

3:01:34

pretty soon. And perfect. We see that here. It should be with the IP address. And there you go,

3:01:42

you can see it cannot access this IP address that is the server IP. So let me copy that. And I will

3:01:50

paste that there notepad, let me copy this IP address, we will go back and inside our database

3:02:01

we will set the server firewall at that IP address.

3:02:08

And we will save that perfect. Now that should not be an issue. Let's hit publish one more time.

3:02:16

This time, it should be able to push all the migrations and deploy our code to Azure.

3:02:23

Once everything is done, it will load the website which is bulky book.as, your website's dotnet.

3:02:32

Great, it automatically opened up and perfect. This looks great. If we go to category,

3:02:38

there won't be any category. But we can create a new category here.

3:02:46

And our alerts are also working. So perfect. With this, we have seen how we can deploy our website

3:02:53

to Azure. That being said, this is only for testing, but the SQL database that we have, we are

3:03:00

getting charged for that. So let me remove all the resources. And we will also delete this profile.

3:03:08

The reason I want to delete is I only wanted to show you how easy it is to deploy your code to

3:03:15

Azure. So with that, let's go to our resources, select all of them and let's delete them. So,

3:03:26

with that we have seen the CRUD operations that we can do with category using MVC in dotnet six.

Azure Deployment

3:03:36

So first of all, congratulations on completing the CRUD operations using dotnet six MVC. We have seen

3:03:44

quite a few things in this course. But right now, we are just getting started with dotnet core.

3:03:51

And there is a lot more to explore and learn when it comes to dotnet core. Even though this

3:03:57

course has come to an end, I have advanced courses that will take you from the basics that you have

3:04:02

learned here all the way to exploring new concept. Let me walk you through some of those concepts.

3:04:10

In the full course of dotnet core MVC, we will take what we have built so far and convert that

3:04:16

to an anterior architecture, because that is a real world scenario. Because that is a scenario

3:04:23

that is adopted with most complex project, then we do not interact directly with the data context in

3:04:30

our main project, we will be using a repository pattern and unit of work to interact with our

3:04:36

database. So we will see dependency injection and how to inject that using a repository pattern.

3:04:44

Then we will further split our project inside areas in dotnet core, which is a folder like structure, but everything gets much more organized. We have seen temp data,

3:04:56

but we will see ViewBag and view data with data attend core and see how all of them are different.

3:05:04

Next we will use with a loads rich text editor data tables with dotnet core and API calls

3:05:11

with any application authentication and authorization leaves very important role.

3:05:17

And dotnet team has given us the razor Class Library, which is identity in the dotnet world.

3:05:24

That makes all of the authentication super simple, because the basic functionality and table structure is already implemented for us.

3:05:34

So we will see how to scaffold that identity in MVC application. Next, with authorization,

3:05:41

we will see how roles play an important role. And how can we modify our application based on

3:05:47

the role of the user that is logged in. In a real world scenario, we also accept payments. So I will

3:05:56

show you how to accept payments using Stripe and also give refunds when the order is cancelled.

3:06:03

Session is one of the key feature that was very helpful, but traditional MVC application. With

3:06:09

dotnet core also session is not gone. We just need to do a little configuration and add session to

3:06:16

our project. And we will see that in action. Most of the applications will send emails in some way.

3:06:25

I will show you two approaches, one using the SMTP server and second using applications and grid.

3:06:33

Then with the modern applications, we have social logins like Facebook, Twitter, and so on. So I will show you how the dotnet team has made it easy

3:06:43

to integrate social logins with the help of Facebook. Lastly, there are some advanced concepts like view component how to see database with DB initializer.

3:06:54

And once the data is seated in the database, our final goal is deploying the application to Azure.

3:07:02

So you can see there are quite a few topics that we will cover in the advanced cores. But with that you will get a solid foundation to build your real world project. And if you'd like

3:07:13

this free course, please subscribe and like this video, and I will have more free content uploaded

3:07:19

every month. So good luck with your journey of dotnet core and happy coding