**Project Transcript**

In this course, you will learn how to create a full stack application using MongoDB for the database, Java and Spring

Boot for the back end and react for the front end. Farhan Hasin Chaudry teaches the back end section of this course. And

Gavin Lon teaches the front end section of the course. Both these instructors are very experienced and have created

many popular courses. The project featured in this course is a great example of the creation of a separation of

concerns between the client code and the server code. By implementing a loosely coupled architecture. These two parts

implemented using different technologies can evolve in parallel and independently from one another. So let's start

learning from Farhan. Hello, everyone. Welcome to the Java with MongoDB course on the freeCodeCamp YouTube channel.

I am for Han has ensured three one of the instructors in this course and I will teach you all about developing a back end API

using Java EE Spring Boot, and MongoDB. The other half of the course will be taught by Katherine lOn. Hi, I'm Gavin

lOn. I'm a full stack developer and have over 20 years experience working as a software developer. In this course, I'm

going to be playing the role of a front end developer. And my responsibility will be to demonstrate building a front end

using React js. The Spy application that we are going to create will use HTTP GET requests to appropriately

retrieve data from a remote server through communicating with the relevant API's endpoints. And HTTP POST

requests will also be made to the relevant endpoints to appropriately save data on the server side. Of course, our data

will be stored within a MongoDB database on the relevant remote server. So without further ado, let's jump in and start working

on our API. In the beginning of the course, you will have to download two new software's from the internet. The first one

being Java Development Kit, or JDK. If you already have JDK

installed on your system, you should be good to go with that. But just in case you do not have it installed, I'd suggest you

navigate to oracle.com/java/technologies/downloads

The link will be in description. Once you have landed on the download page for Java, you may find multiple versions of JDK

available as of the recording of this course there. The latest version is Java 19. And the latest LTS version is Java 17.

lts C stands for long term support and JDK 17 will receive

updates until September 2020 For

JDK 19 On the other hand, will receive updates only until March 2023 and will be superseded by JDK 20. In many cases, when you

are deploying a Java application on a production server, it's suggested that you use LTS release for better stability.

And for this course I will stick to Java 17 Since that's the latest LTS. Now depending on when you're watching this

course, there may be a new LTS release so feel free to download and install that. So I've switched to Java 17 And since I

am on Windows, I will go to windows and the 64 bit installer. As you can see, it's only 150 2.85 megabytes and

depending on your internet connection, you it may take a few minutes to download.

Once you have downloaded the installer, make sure you have installed it on your system like any other software you have done

before. And since the instruction is so straightforward, I will not spend a lot of time showing you

the entire process. After you have finished installing JDK on your computer,

open up your terminal window and write Java C space double dash

barsha. And if you don't get any errors from this command, this is a

positive sign and as it says Java C 17 point 0.5 which indicates that I have successfully installed seven

point 17 point 0.5 version of the JDK. Now Java C is the short for Java compiler which you may or may not have already guessed.

Apart from this command, you can also use Java double dash version to check the version of your Java Runtime Environment

and as you can see it says 17 point 0.5 lts which means we are good to go.

Since you have successfully installed the jdk on your computer Next up you will need a good code editor or IDE. When it

comes to Java, there are plenty of options out there. You can go into Eclipse or maybe configure Visual Studio code. But I would

suggest that you go with IntelliJ IDEA. Now the people at

Jet Ian's are known for making really good IDE s and when it comes to Java IntelliJ IDEA is pretty much the industry

standard. Although the ultimate version can cost you some money, there

is also a free version available by the name of Community Edition. Or to download IntelliJ IDEA, navigate to

jetbrains.com/ideas/download. Again, the link could be in description and look for the Community Edition. This is the

flag download button here. And depending on your platform, ie Windows, Mac OS or Linux, you may get a different format. For

example, if you're on Mac OS, you will get a DMG image. And if you're on Linux, you will get a tar file. So I'm on Windows and

I want to download the exe installer and not the zip one. So I'll select exe, and then hit download. Now the IDE is a bit

larger than the JDK in size at 649 megabytes. But again,

depending on your internet connection, it may take a few minutes to download. Once you have downloaded the software,

please go ahead and install it like any other software that you have done before.

Congratulations on you're getting the JDK and IntelliJ IDEA getting installed. Next up, you will need an account at

MongoDB Atlas. Now you may or may not already know that MongoDB Atlas is a platform from MongoDB themselves that allows

you to spin up MongoDB servers on the cloud.

So in order to use MongoDB Atlas, you will have to maybe get to mongodb.com/atlas/database.

Again, the link will be in description and use the Sign In button.

Once in the sign in page, you can either use your Google or GitHub account to log in or you can sign up with your other

email addresses for a new account. Now, I already have an account on MongoDB Atlas. So I will just use those credentials

to log into my account off screen. As you can see, I have successfully logged into my account. And For first time

users, you may have to set up a new organization of sore just

just put any name as the organization name and you should be good to go. Since we are using these for just our

development purposes, it's okay to just use any data that you have. So once you have set up your organization, you can

create new projects. So I will hit the New Project button here.

And I have to give my project a name. Now since you are creating an API that is related to movie I'll just call my movie API. And

hit Next I will just go ahead and create project since I don't have any

other member to add.

Extra now we are inside our movie API project. And we are ready to create a new database. So you can use the builder

Database button here to create a new MongoDB database on MongoDB Atlas. So just go ahead and click this button.

And there are three different categories there is server list there is dedicated and there is shared. Now for development

purposes, our shared server should be fine and it's completely free. But if you're working on an application that

that has some user and have some real life usage, you may go with

dedicated Serverless is something else and maybe we can talk about it in a different course some other day. So select

Share, and hit create.

Okay, next up, they will ask you about which provider you want to use, I always go with AWS. Since AWS has a server on Mumbai,

which is near Bangladesh and has the lowest ping for me, but if

you like you can go with Google Cloud or Azure. Now select whatever region is the nearest to you to make sure that you are

getting the lowest amount of being possible. Next, you can choose a dear.

Again, for free usage, we use the aim zero sandboxed here it

will give us 512 megabytes of RAM which is more than enough for our usage we will have a very small amount of data to be

honest. And then next step there are additional settings you don't have to touch them and the cluster name. Now you can

customize the cluster name if you Wish but I will just leave it as cluster zero since I like it

that way, then hit Create cluster

Okay, next up, you will have to create a new user for this cluster. You can also use an SSH certificate, but I will just go

with the old trusty username thing. So I will name my

usernames Solid Snake since I'm a Metal Gear Solid fan, but you

can name it anything you want. Now, when it comes to password, make sure you are always choosing a very strong password

and not ones that are easy to remember. Use a password manager

maybe but always use hard to guess passwords. So I will just use the auto generated secure password.

And I will also make sure that I am copying my password to the clipboard, since I will be needing this later on. But then

hit the Create User button. It shows Solid Snake and the password.

And finally, you will have to set up some IP access list. So the way it works, once you have set up a new database, you will

have to allow a set number of IP addresses that can access that server. You know opening up a database server to the world is

not a good idea. Security wise, so you should always use your

current IP address here. But since this is just a course, and we are using this database

for development, I will open up the server for all the IP addresses out there. And to do that, you can write 0.0 dot 0.0

slash zero and in the description, you can just write

a view and hit Add Entry. See that that's there. And then click on

finish and close. Okay, they will congratulate you and whatnot Hi quickstart guide

Yes, go to database

disclose it. Excellent. Now we have a database up and running on

MongoDB Atlas. And we can use this database ganache to eat read and write data from and to eat.

And using just like a local database that you may have used in your computers, you see MongoDB Atlas is a very

beautiful service. I have used it for development as well as for production in the past and I have never had any bad

experiences with it. So next up, you can use some sort of database client to

connect to this database, maybe something like MongoDB compass,

so you can go ahead and click on Connect. And MongoDB will give you multiple options that you can

choose from. So first, you can connect to your database server using MongoDB shell, you can connect to the MongoDB server

using your application that you're working on right now, you can connect to MongoDB server using MongoDB compass which is

MongoDB own graphical user interface for working with databases and I will just show you that. Finally you can also

connect using VS code. And let's be connect using MongoDB compass for now.

And I do not have MongoDB compass on my computer. So I

will be Windows 64 bit

and I will just hit download compass. Once you have downloaded and installed MongoDB compass on

your computer, you can start the software from your start menu or desktop. Once you start MongoDB compass it looks something like

this where they are giving you the option to connect to a new

database by using this kind of URI. Now where do you get this

URI. If you come back to MongoDB Atlas, you will see that they

are actually giving that URI right here. Just use this copy button to copy the entire URI and go back

to MongoDB compass. Now remove this localhost URI from here and

paste it yours. Now as you can see it says MongoDB plus SRV

Benue username, followed by a colon and then it's looking for

a password right. So what we are going to do, we will use the password that we created earlier and paste it in here. And then

we'll leave everything as is and we He can say Save and connect. You can name it anything. So I'm

naming it V Avi, you can change its color if you want to, and say Save and

connect. Okay, so it took a few seconds. And as you can see, we have

successfully connected to our moving API cluster. Now, as you can see, there are two databases, one of them is called

admin. And the other one is called Local. Now, we do not have to do anything with these, these come by default with all

the clusters out there. So just leave them be and we will create new databases very soon. To create a new database, you can

always go and click this plus sign. And inside database name,

let's put b What really, let's call it movie,

API deep. And inside collection them. Now if you're new to databases, like

like MongoDB, or document based database, you may or may not

know that collections are like tables in case of relational

database systems. So in something like MySQL, or Oracle database system, you have databases, and inside those

databases, you have tables. Now, in case of MongoDB, and other

non relational databases, you have the database name first, and then collections. For example, in our cases, we will

have a collection of movies where we will save information about different movies. So we can say movies and in create

database. So as you can see, we have a new database sitting here movie API dv, and there is a collection called movies. So

right now, there is no data at all.

Now in order to work with our API, it would be convenient if we have a bunch of data in our collections. And if you navigate

to github.com/ach, si n ch y, which is my username and slash

movies, you will get the entire code for this API already written. And inside that repository, you will see a

folder that says underscore data. Now inside this folder,

there is a JSON file that you can use as a source of data for

the collection we have just created. So I would suggest you go into this JSON file, then select raw,

and then hit CTRL S and say this JSON file somewhere on your computer.

So once you have downloaded the file, you can then import it to your collection. To do so, come back to MongoDB compass. And

while you have the movies collection selected, you can click on this Import Data button or this Add Data button. Then

select Import File. And our format is JSON. And we'll select

the file from our downloads and movies dot JSON. And you can

also take a stop on errors, then hit import, wait a few moment.

It done. And viola, we have the information of n different

movies on our database. So as you can see, each of this movie is a document Okay, as it as it says documents

then. So inside the collection, each of this movie are a single

document. And what do we have inside these documents? Well, we have an ID we have the IMDB ID for this title whose inputs, we

have the release date, which is a string, we have the link for a trailer on YouTube. Then we have an array of genres. So this

particular movie belongs to five genres. Then we have a link to

our poster. We have linked to end backdrops that will be later utilized in

the front end application as you can see, finally, we have an

empty array for review IDs. Now, this will come into play later on in this course, so just be patient with me for now.

Okay, so one thing that I'd like to mention right here I am using the tmdb.org API for this

images are TMDb is an excellent website. And they provide a free

to use API for developers as long as you are giving them credit. So I do, so I will be using the API for portraying the

images. So now that we have created a new database on MongoDB. Plus, we have set up a new collection, we have imported

some data into it, it's time for us to start working on our API

code. Now the first step to writing our API using Spring Boot, is to

initialize it first. To do that, navigate to start.spring.io.

Again, links will be in the description. And let's start initializing our project. First, we'll have to pick a programming

language, which is going to Java in this case. But if you're familiar with groovy or Kotlin, you may go with them, then we

will have to pick a project autumn build automation tool. And the initializer supports both Gradle and Maven which are

not the most popular build automation tool for Java, I am doing with Maven, simply because I have more experience working

with this then Gradle. Next up, picking up the Spring Boot version 3.0. Point one is the latest stable version. And we

will be using that throughout the course. But if you would like to try out some more bleeding edge features, you can

use the snapshots once you have become more familiar with the framework.

After that, we will have to fill up the project metadata.

Now the group is usually written in reverse domain format. So for

example, my website is Farhan dot Dev. And I would like to

name my group date dot for her. So if you do not have a domain

of yourself, you can name it anything you like, as long as that package group is not referring to someone else's

project nextra that artifact name, the artifact name is simply the name

of the project. So in this case, I will be putting movies in here, but feel free to call in something else if you feel like

that. Next there is a description field and you can put something

like

after that, we will have to pick up packaging for now java is a very popular packaging format for Java and it stands for Java

archive wire. On the other hand, it stands for web archive. And you may think that will wire is more suited for this project

since it's going to be a web application, right? Maybe yes, but since jar is more widely used than where I will be going

with. Next speaking the Java version, obviously, I'm going to pick 17,

since we just spent quite some time setting up Java 17 on our computers. Once you are done with the left column, let's

focus on the right column. In this one, you will have to pick

the dependencies that you would like to use in your project. So use this add button, and you will be presented with a long,

very long list of dependencies. And the first one that we are going to use in our project is Lombok, it's an additional

library that saves you from writing a lot of boilerplate code. And you will see what I mean in action very soon. So

please be patient. Since we are going to pick multiple

dependencies, you will have to keep pressing either Ctrl on

Windows or Linux and command on Mac. In fact, it says here, right on the right corner press Control to multiple apps. Okay

for multiple ads, so I'm pressing down on control and I will pick Lombok

Lombok has been picked. Next up spring web.

And finally, let's see.

No nada SQL under no SQL, we will need Spring Data MongoDB

Yeah, that's that's pretty much it. You can also search for

things here, for example, dev tools, and let's add that as well. Once done, you can exit out of this selector by pressing

Escape on your keyboard and have a look at the list of dependencies and make sure you have the

Everything. Now there are other dependencies that you may learn about later on, such as the reactive web,

or Graph QL, or rest repositories or something like

maybe spring security, which can be used for authentication and authorization. There is more to client if you're into auth, and

things like that. But since we will not be getting into authentication and authorization in this course, we will stay out

from the get go. Once you are happy with the configuration in this initializer to go ahead and click Generate.

It will take a few seconds to generate the SIP archive, and it will start downloading automatically. Once downloaded,

you have to extract these movies dot zip or whatever you have named your artifact dot c file to somewhere in your computer

easily accessible. After you have extracted the downloaded zip file somewhere on

your computer, you can go ahead and open it in your favorite IDE or code editor. In my case, it's IntelliJ IDEA and I have

extracted my project on my D drive. So I will go ahead and click on open first.

Then come to the folder where I have extracted the project and I

will pick it up. Okay, so IntelliJ IDEA will ask whether you actually just this

project or not. This is a safety feature to save you from malicious code from running on your computer. So since we are

writing it ourself, I just pick trust project.

And give ideas some time for resolving all the dependencies.

And once it's ready, you are free to go.

Let's try to understand the project structure first. So ignore the dot env and folder, it's managed by Maven. So you

don't have to worry about it right now. There is the src folder, which is short for source. And it will contain your

program source code as well as the test code. So we are not going to write any tests. So let's jump into the main folder.

So inside main there is Java and resources, we will have to work inside both of these directories. So let's go inside

Java. And as you can see, here is the name of our project after 400 movies, it it may be different in your computer

depends on what you have chosen in the initializer. But it should be something similar.

Then there is a class that represents your applications names. So since I have named my artifact movies, it says movies

application, which is pretty appropriate to be honest.

In other movies application class contains around 10 lines of code minus spaces in between. So let's try to understand what

it's doing. The first line is actually declaring a package which is pretty self explanatory. Next, there are two

input statements. The first one is a class called Spring application. Now this particular class contains a method called

Run. And to start your Spring application, you actually have

to call this run method, and then pass your application class to it. And then you can also pass some command line arguments

which we are not going to touch the second import a statement imports like the Spring Boot

application annotation. So annotations in Java are usually used to let the compiler know about what this class does.

So other than this class, there is more or less nothing inside this package. So let's try to run our application and see if

we can we can have anything on our browser. Okay.

So you can actually click on this play button, besides the main method to run the program or this class play button, or

this play button at the top. Let's use this one.

Okay, so if your IDE says Lombok requires enabled annotation processing, just go ahead and click on Enable annotation

processing and it should be fine. Now, if you look at the terminal, you will see that

there is a nice arc that says the spring the springs version and there's bunch of logs and

There is an exception. That exception states that the application is failing to connect to any MongoDB database

at all. Which is true, since we have not let the application know about our, our cluster, our username, our passwords and

things like that application cannot connect to our MongoDB database. Now, it's fine if you want to leave this error here.

Or if you want to get rid of this error, you can just first stop that application.

Open the POM dot XML file, which contains a list of all the

different dependencies that we are using. And then look for

MongoDB.

It should be here somewhere yeah, there you go. Dependency artifact the Spring Boot starter data MongoDB. So just highlight

the entire thing. And press ctrl and the forward slash which will

comment this out. Then right click on form dot XML

and then go to Maven and go to reload project

now try to run the application once again.

No your whatsoever. So if you look at the logs, it will say that Tomcat started on

port 8080 Tomcat is actually an web server that will serve as

the web server the development web server for our application. Now, you'd have to switch to a web browser to see the

application in action. So let's do that.

And then you can navigate to localhost

88080. There you go. Now it says whitelabel error page, that's

fine. At least we know that the application is booting up and we are actually hitting the API itself. And not just some other

page when what is zero states is that there is no endpoint at

all. So although we have an API, there is no API to hit. I hope

that makes sense. Now let's go back to our code and try to write our first endpoint.

But after where it says the Spring Boot application,

we will say Rest Controller.

It's another annotation that lets the framework note that this class is actually a REST API

controller and not just another class. Then inside this class,

you will have to say get mapping.

And next we'll have to create a new method called public. And the return type will be string. And the name of the method can

be anything but let's just call it API index for now. Because this is the

route endpoint or or list, let's call it API route.

And we can say return.

Have no or I know have our programs are boring, but it's essential, you

know. So this gate mapping and notation to each, what it does,

is that it lets the framework know that this method right here, API route, is a good endpoint. So if I put a bracket

right here, and then inside, I'll say slash just a forward slash. Let's stop our application and run it again.

Let's see if there are any errors. looks fine to me. Let's switch to our web browser

and do a refresh. As you can see, it says hello world. Let me just increase the zoom level a bit. So you can

tell alright. Now instead of putting a forward slash here, if I say for say

slash route, rerun the application

you will see that the whitelabel error page is back. But if you navigate to slash route, you will see

Hello world. I hope this makes sense. Although DCS are valid

eight point, we are not going to make our API like this, we will actually divide our application into separate layers, there will

be a service layer, there will be a data access layer, and there will be an API layer. So let's go forward and get rid of

this code and restart to put together our movies API.

Now, in my opinion, the first logical step would be to configure the database for our application so that we can

successfully connect to the MongoDB cluster that we have set up previously, and talk to it.

To do so, open up the src folder right here, go inside main,

go inside resources, and open the file that says application

dot properties. Now, in this file, you will have to write the different

application properties that is related to your project. For

example, right now, we are trying to configure our MongoDB

dependency. And to do that, you will have to write

spring dot data dot Mongo DB dot database

equals to the name of the database. I hope you remember that we named our database something like movies API dB. So

we will write movies API dB, and next spring

dot data dot Mongo DB dot URI. Now in here, you will have to

actually write that URI to your MongoDB cluster like you did

with Compass. I have my URL already copied to my clipboard. And I hope that

you have saved it somewhere safe. Like I said in the beginning of the course that I will just go ahead and paste it

right there. And you can get rid of this slash test thing. And the last

slash, and it should be fine. Okay, so our application should be able to connect to our database. Now

to test it out. Let's go back to POM dot XML. Let's remove the

comments.

And then let's start our application. Oh, see, we have forgot to reload our project. And the IDE

is actually suggested me to load the map and changes. So I can just click this button or you can go ahead and right click and

go to Melbourne and reload project. Let's use this button and see if it works fine or not.

Of course it does. Let's run the application.

And now the Euro is called. And actually it says some things

about the MongoDB driver cluster. And this is a positive sign. If it failed to connect to the database itself, it would

have screamed at us and whatnot. But thankfully, it's fine.

Now, if you have worked with any applications, specifically web applications in the past, you've may already know that putting

sensitive data like this in a file that's going to be in your GitHub repository is not safe at all.

Instead, what you should be doing is putting all these things in a dot env file. So we can right click on Resource, go

to new and create a dot env file for ourselves.

And we can write down a few properties here.

Mongo database, then Mongo user,

then Mongo password, finally, Mongo

cluster. Now you can go back to the properties file, and you can

start moving these values one by one to your env file. So this is

going to be the Mongo database. Next up the Mongo user name which is

Solid Snake in my case, good among but then Mongo password.

Finally, the cluster.

And I sometimes like to put them inside quotes, this makes sure

that if one of the passwords or username or clusters contain some special characters in them, it would not

disturb the order of things. So I like to put this quotation

marks around each fabs. Okay, now we have our env file ready to go, what I would suggest that you create another env file,

simply called dot env, dot example,

for later reference references

then make sure you open up the dot Git ignore file. And as you can see, the Initialize project actually comes with a pretty

good Git ignore file already. But we'll have to add a few things for ourselves, right. So you will say, you can actually

copy this thing from here and paste it and say, in the white.

Yeah. And then you say.in With this will make sure that you do

not accidentally commit that dot env file into your repository and open up your database to the word.

Now, the only problem with this spring is that spring doesn't support reading dot env files out of the box.

So what we'll need is we actually need to install

a new dependencies to our project. And I would like to take this opportunity to show you how you can add new

dependencies to the project later on. So I'll switch back to

my browser. I'll open up a new tab and I will search for

Maven dependencies, something like that. Yeah. And here is Maven

repository. Then let's search for something along the line spring. Dog

Ian's.

Yeah, I am referring to this one by Paul Suarez. I apologize if I

have pronounced his name wrong. Let's select this one. And we can see that it says this spring

plugins 1.0. And it says Central and whatnot. Okay, so the name of the thing is made up Paul Torres spring dot EMV. So as you

can guess this is his group, group name and thesis, the name of the artifact or the or the project itself. So we can go

back to our code. Let's open up POM dot XML. And inside the

dependencies bar, make sure you are inside dependencies and not outside somewhere here. After the last dependency, we will

create a new one. So it is a dependency.

And the ID helps us by auto completing this. And we will say

that the group ID is mi dot Paul

is he h, w A are set just to make sure you can always copy

from here. No one's going to judge you for that. And then name of the

artifact which is spring dot E and now it's not strictly necessary to define a version as well. But

you can do so by saying version and then put maybe the latest from two point 5.4

by one. Yeah. So that's it, let's reload Maven.

And yet now our project has the ability to use the spring dot env artifact and three dot env fights. Let's go back to our

application properties and replace these hard coded values with our dot env references. To do so open up the resources and

we need the env so Mongo database Mongo user Mongo password, okay?

So you will first have to get rid of this value and then start with $1 sign

and curly braces. And inside the set of curly braces, you will see say e NV dot

Mongo. What do they call it? Database?

Yeah, Mongo database, and you can kind of copy it from here then

replace Solid Snake by Mongo user.

And then they should be Mongo password. But sore tricked.

Yes. And finally a cluster.

Cluster Yes. Does this mean? Okay, we can ignore this typo.

And if you want Singapore, you can actually come to this problem step and have a look at what are what's the ID is

complaining about. So the annuals property, let's just ignore them from now.

Which ones per year? Yeah. Okay, let's ignore these for now. And the typos. Okay, Mongo is not a typo.

Except let's try running our application once again and see if our env file has taken any effect or not.

So as it turns out, our MongoDB connection is fine, which means the end file has been read successfully. And we have

successfully secured our confidential information from going out into the world. Now you, you don't have to write

everything inside an env file, you can actually use this en v dot name of the variable notation to access any

environment variable variable in your operating system. But I will not get into that anymore since we don't need it. Let's go

ahead and get along with our project. Now in our application, there are two types of data that we

are going to work with. First there are the movies and then there are the reviews. So we will need to first create two

separate classes for representing these two entities in our application. So first, let

left click on your project name new and Java class let's call it movie.

And inside this movie, we will first have to annotate this class as a document.

This will let the framework note that this class sheet presents

each document in the movies collection. And we will also define the name of the collection here movies.

Then inside here we will have a bunch of private data. So

private object ID ID this will represent the ID of this movie and private

IMDb ID sorry,

string IMDb ID which will represent the IMDB ID of this movie. And

preferred string, title,

private string release date

for private string Schadler link this will be a link to the YouTube page for the

trailer private is string poster a link to the image on the internet

private list string this will be the genres because there can be

multiple genres of a single movie. And these lists class

need to be imported at Java util by any private list string

and this will be the backdrops now this backdrop images will

actually be used in the front end application as you will see later. Let's just quickly jump to MongoDB compass and make sure

that we have indeed written all the necessary properties for this class.

So as you can Let's see we have written high D IMDb ID title yes released a trailer link genres poster, backdrops, air review ID. So you

you, we will add review IDs, a few moments later.

Now, we will have to also annotate this ID field

as an actual ID, this will lead

the framework know that this property should be treated as the unique identifier for

each movie inside the database. Okay, apart from this we will

also need getters setters that do is string methods and some

constructors. So instead of writing out getters setters to a string for each of these private properties, we can just say

at the read data, and this is one of the annotations that comes from the Lombok project. And it takes care of all those

different gators setters and two string methods. Next, we'll also

add all args constructor, this is basically an addition for

creating a constructor that takes all this private field as

argument finally, no args constructor, which means another constructor that takes no

parameters whatsoever. Okay, next, let's create the reviews. So Java class

reveal, it's the same deal. So this will be document

and the collection name will be G fields. Now, we haven't created this collection yet. But the application itself has the

ability to create new collections. So we don't have to worry about that. We'll add data just like before.

All args constructor, Nords. constructor, and then

there will be two private properties for a strong private

object ID. ID, this will be the ID in this case.

And then private

review body just just body string body.

Okay, so as you may have already seen in the database design, that each of these movies has an array of this review, and we

haven't yet added this this area of reviews to the movie model yet. So to do that, we will go back to the movie class,

go near the end and say private

least review. Review IDs. Okay, so it matches this name review IDs, and the

hall that our names are matching as well. Yes. Now if you write it like this, this will be an embedded relationship. So all

the reviews that are related, this movie will be added to this

list of reviews. Now this is completely fine. In fact, if you are modeling a one to many relationship, such as one movie

can have many reviews, this is the way to go. But since I'm

trying to demonstrate some of the features of Springboard MongoDB data, I will show you a nifty annotation right here. So

I will say document reference. So this will cause the database to store only the IDs of the

review. And the views will be in a separate collection. So this

is called manual reference relationship. And there are some other ways to

create relationships in MongoDB. I would suggest that you search them up on Google or MongoDB sufficient documentation and

read about them on your leisure. It will give you a lot of context. So now that we have our movie and review models ready to

go, we can go ahead and start writing on controllers. So I'd

right click on my packet then a Java class. This will be called Movie

controller. And this is our first official day API controller that we are writing in this API. Previously,

we wrote a simple controller in our movies application class,

but this time, it's going to be a bit different. So first, we will need to annotate this class as a Rest Controller.

And instead of mapping it to localhost 8080, I want to make

it sorry. Mapping request mapping, I want to make it map to slash API

slash v1 slash movies. So any requests to slash API slash v1

slash movies endpoint will be handled by this particular controller. Next, I will create a new gate method. So I will say

get mapping. And then public.

Let's return string for now.

All movies, and then we can return

all movies. OK, let's try to run our program and see if it works or not.

Let's hit the play button.

Okay, it's compiled fine. Let's go back to our browser. And here

localhost 8080 slash route. We will get rid of fruit and say slash API, slash V one, slash

movies. And it works. Now although it's okay to return these strings

from this endpoint, it's better to return a response

entity I'll show you why in a minute. So this is a generic pipe. So we'll say string.

And then instead of returning or returning, and instead of returning all of this like this, we will say,

new response entity of type is String. And this is the value.

Then HTTP status dot O.

HTTP status dot okay means 200. Let's read on this program.

Refresh, and it works just as expected. Now, you may not see any difference there. But if you go back to your terminal

and say, curl, I HTTP localhost 8080 slash API slash v1 slash

movies, you will see we are indeed returning the 200 response code.

This comes in handy when working on a front end application and any REST API out there should return proper status codes.

Since we have the movie model right here, let's try to use it and pull some data from the database.

For that, we will need a service class as well as a repository

class. So let's create the repository first, because that's the shorter one. So it's a Java class. And we'll select

interface because repositories are of type interface. And then we'll say movie in repository.

Yeah, now this interface will extend Mongo repository. This is the generic type. And we will have

to let it know what type of data we are dealing with. So movie, and we'll also need to let it know what type of ID we're

dealing with, which will be object ID, there you go. That's all for this interface, you will need to

unnoted this interface as a repository. So that framework knows that this is a repository.

And then let's create a service class which will be movie

service. And this class will not extend anything, you will just need to

annotate as service. Now, inside this class, we will write the database access

methods. So the first one will be public

Get on movies.

Let's just name it all movies. And let's just change our

controller method name to get on this because since it's a get

mapping, we're saying get all modes. And the return type will be list

of what will return a list of movie from this method.

Now, inside the service class, you will need a reference of the repository to do that you will say

movie repository it will be a private field.

And its name will be movie repository. Now if you know Java, you may know that you will have to first

initialize this bit of code, you will either have to initialize

it using a constructor, or you can just use auto wire an

audition what this will cause it will let the framework know that

we want the framework to instantiate this class here for

us, okay. Let's go inside the method now. And we will say return

move the repository dot find all these find all method is described inside the Mongo

repository class. As you can see, it says right here find all

and it will return and least of the data type that we have passed right here certainly stock movie. So that's our

service class right there. And then let's go back to our controller, we will need a reference to our service class.

So we will say private movie service movie surface. And just like we did in the service

class, we will auto wired the service class. Since it has the service and audition right here, we will set auto wire.

Excellent. And it's also now inside the good old movies. Instead of returning a response

entity of type string change it to list

so we are returning a response entity of type list movie.

Then inside the method, we will have to write return

movie service.of movies

new response entity of type list

movie and inside movie service drug called movies comma HTTP status

dot okay. Yeah, let's take a look at our repository. Once again, it's all

set up. Services fine, the controller is fine. Let's cross our fingers

and see if this works or not.

Going to go back to our browser and hit refresh.

And the reason behind this problem lies in my EMV file. So

as you can see, we have called our database movie API dB.

But in our configuration, we mistakenly put an S right here.

So we'll get rid of the s. Res restart the application and hope for the best.

And yeah, it works. As you can see, there are 10 movies in this list. Since it starts at zero and ends at nine. There are 10

movies in this list and the code is perfectly fine. Now let's go

back to the code and let me re iterate on what we have done so far. So in rest API's, usually there are multiple layers. So

one of the layers is the API layer, which is this controller and it will only concern itself

have about the task of getting a request from the user and

returning a response and nothing else. And that's what it's doing. All it's doing is it's using a service class and

delegating the task of fetching all the movies from the database. And give me giving it back to the API layer. So it

calls the all movies method inside

the service, gets the list of the movie and returns them with

HTTP status. Okay, it doesn't know what's going on inside the

service class. Now, when we come back to the service class, this is where most of our business logic will go.

For now, there is nothing to be worried about in this classes. But soon, we will work with a little bit more complex business

logic in this API. So the service class does is it uses

the repository class, and talks to the database, get the list of the movies and returns to the API layer. Finally, the

repository layer is kind of the data access layer for our API II

does the job of actually talking to the database and getting the data back.

I hope that makes sense. Now, since we have a list of all the movies in our databases, let's complicate it a bit more by

trying to access a single movie. Now, we will begin by creating a

new get mapping. And that is

get mapping and this will actually take an parameter.

And we will have this a slash that inside a pair of curly braces will say ID. So we are trying to search a movie by its

ID. So we can set public response entity. This time, we will return a single movie.

And we'll say get single movie. Sorry for the type book.

We'll close the pair of parentheses

and we'll start our method. Now inside the pair of parentheses, we'll have to say

PATH variable, which lets the framework know that we will be

passing the information we got in the mapping as a path variable. Or we will be using the information passed in the

PATH variable as a string,

or just object. Id ID.

So what this does is it lets the framework know that whatever we are getting through this path variable, we want to convert

that to an object ID called ID. And then we can say

return first we'll have to go back to our service layer and write a new method here.

Public movie single movie

and we will say return movie repository dot

find by ID and we will get that ID right here.

And it goes here it David and the squiggly line it says that whichever option okay?

So the thing here is that the find by ID method may not find

any movie at all, maybe the ID you have passed doesn't exist.

So in those cases, you it will have to return now, so we'll have to let Java know that it may return null by saying

optional movie and we will just import the optional class

we will also have to fix our response entity movie will say

optional. Movie

optional has been imported nice and then We can say return new response entity

optional movie. And inside there we can say movie service

dot single movie, we will pass the ID that we have received

comma HTTP status dot. Okay.

Let's restart the application

let's go to our MongoDB compass inside the movie collection and

copy one of the object IDs right here let's copy

for Roald Dahl's Matilda the musical. Let's copy the object Id

go back to our browser.

And we can say movies, slash and then the object ID.

As you can see, we have found Roald Dahl's Matilda the musical. Now, the problem is that I do not want to expose the

object IDs of my collection entities to the public. Instead,

what I want to do is use this IMDb ID to search for new

movies. So let's see if we can find a way to do that.

Now although the repository comes with built in methods for

searching with ID, it doesn't come with methods for searching with IMDb ID so what we'll have to do is we'll have to implement

that method ourselves. And it's really easy because there is

something called automatic queries that lets you

farm where it is dynamically from.

property names so we will say optional.

Movie because it may return null once again, we'll have the input

optional movie and we will call it

find movie by

in dB ID and it takes a string

I am DB ID. Now just by naming this method find movie by IMDb ID

MongoDB data Spring Data MongoDB will understand what we are

trying to do. It's that much intelligent. So now we can go

back to our service class. We are no longer getting an object ID but we are getting a

string IMDb ID

and instead of find by ID we will say find a movie by IMDb ID, IMDb ID

and that's it. Let's go back to controller.

We'll change this from ID to IMDb ID by the way you do not have to name the ID IMDb ID exactly in this method. It's

It's just that I know what I'm doing in a later date. So we

will replace object ID by string

IMDb ID and then response entity optional movie. This is fine.

Yeah IMDB. So technically, this should do our job. Let's check it out

let's see seems like there has been a mistake.

Let's go up

it looks like there is a problem. Let's go back to

our movie repository.

And the music It is it should be IMDb and not IMBD. So as you can see, the

framework is clever enough to see that I have misspelled

the property name right here. So let's run the program once again and see if you're trans or

not fine tuned by cannot find

up here, I have to correct terrorism,

DIA. And yeah, it shouldn't be fine

let's go back to our browser. And as you can see API slash var slash movies. And then I have used the IMDB ID for this movie.

Hit enter, and it works just fine. We can use the IMDB ID to

search for movies. Now, you can actually form dynamic queries like this using

any property name in your model class.

As long as they're unique, because if that's not unique, you will get multiple

movies with the same ID or same name. Okay.

Now that we have our movie controller sorted out, let's start working on our review controller, where we will let

the user send us review. And we will add those under a movie

name. So let's first create a new repository.

View posit Ori

and it spends Mongo D. B,

Mongo repository and the die will be revealed.

And object ID is our type. Yeah,

yeah, that's their next let's create a new service class. Oh, I almost forget to

enter this as repository. So, create a new class called

the views service. And this will be a service class. So I'll go service

here I will say public then

this method will create a new reviews so it will return a new review then here she view

by

creative

then this method will take two parameters. The first one will be a string

which will be the reveal body and the since

the second one will be string IMDb ID so what we are trying to

do here we will first look for the movie with the given IMDb ID. And then we will create a new review and associate that

review with the found movie. So let's first create a new review.

Review few quotes from new review.

Now you may now you may remember that in the review class, we have an pol

arcs constructor and an aurochs constructor. But since IDs are auto generated, we cannot pass an ID to this class. So what we

will do we will also generate a custom constructor that takes only the body. Now in IntelliJ IDEA you can right click inside

a class go to generate and then constructor and we only need the

body part of this class. So that's

now let's go back to our service class and new review and we will pass the review body

Excellent. Now that we have a new review, we will have to associate these

to one of the movies

now that we have a new review object, we will have to insert it to the database and for

After that, we will need a reference to our

review repository. So review repository view repository and needs to be auto.

Fine. And we will say, review repository

dot insert, and review. X. And that's the review taken care of now we can have to

associate with this with one of the movies, right? To do that,

we need a few things. Now the first thing that we need is called a

template to create a template with set private Mongo template,

Mongo template. Now, you have already learned about repositories as one of the ways to talk to the database. The

other way is using a template. You see there are times when a

repository just doesn't cut it, maybe you have an operations so

complex that it cannot be implemented within a repository, or even if you can implement it within a repository, it will be

not suitable. So what we need, we need a

template, you can use this template to form up a new dynamic query and do the job inside the database without

using the repository. So we will alter this.

And then down here, we will say Mongo template

dot update, since it's an update operation, and we need to say

which class to you do we want to update. So we want to update all

of the movies. And then come to the second line and put a dot here. And then as

you can see, there are multiple operations that we can perform. So we want the match.

Now inside the matching of version, we will need a new criteria. So we'll set criteria

dot fire. And inside where we will say IMDb ID, make sure it's matching

up with the column name or the property name inside the database

then.is the IMDB ID that we have got.

Then come to the next line and put a dot once again and then we'll say Apply.

We'll say new update. This is called an update definition

dot push

review IDs and then dot

value sheet view. See, that's pretty much it. Okay, so what we are doing is we

are using the template to perform an update call on the movie class. Because I hope that you'll remember that each movie

in our collection contains an empty array of review IDs. So

what we need to do, what we need to do is we need to update this array and push a new

review ID into this. So we are saying movie dot class then we have to perform the

matching. So which movie are you updating? We are updating a movie where the IMDB ID of the movie in the database matches

the IMDB ID that we have received from the user. Okay,

then we want to apply this update. To do so we call apply.

And then we create a new update definition which does the job of

making the change inside the database. So we say Update, push

and review IDs. So we want to update the review IDs in this found movie and the value of this movie will be revealed. So

that review that we have just created it will be pushed inside the review IDs array okay.

Now, finally, we will have see the First, to make sure that we are getting a single movie and we

are updating that. Now one thing that I have done, which I shouldn't have is we cannot new up a review like this.

Rather, what we need to do is we will just save the review on

insert, because when you call insert, it actually returns the data you just pushed inside your database. Okay?

First, finally, we can return that view we just created.

So let's review the Create review method. We are taking a new Mongo template, we are using it to update the movie with the

new review, we just pushed using the repository. So likely say sometimes you will have to code up complex business

logics by yourself. And as you grow as a software engineer and an web developer, you will start to understand all these things

all by yourself. So let's go back to our review service. Yeah. So I mean, our

review is is fine. If you're in positions where you feel service, yeah. Now let's create a new review controller.

And inside this controller, we will have only one post method.

But you may think, why only a single post method? How are we going to get all these reviews that are associated with a

movie, I'll show you how. Now let's begin by annotating it as a Rest Controller.

And then of course, we will have input request mapping

which is slash API slash v1 slash

mobile movies

Okay, since the review form will be inside a movies Details page, or in the

page where you are viewing a single movie so we can make the requests to the movies and point instead of creating a new if we

use NS N bar. Now when it comes to creating endpoints like this, it really depends on the on the developers preferences, you may

you may choose to organize your API differently from what I am

doing here. And in a real life project I maybe I will do

something different but since it's just for learning, and we are just getting started, I think it's fine.

Inside the controller, we'll first need our service. So we'll say private

reveal Service Review service and we will auto add this

nice then we'll create a new post mapping

and then a new method of Lake will again return a response

entity and this will be of type review. And we will call it

create review method. Inside this pair of parentheses we will say

request body then map

string, comma string and we will call it payload

so we'll import map. So, what we are saying to the framework here that whatever we

get as the request body we would like to convert it to a map of

the key string and value string and we want to name this map as payload you will see why this is

then we will say return new response entity.

And yeah we can say review this review and then it says Service Review service dot create review and we

will pass two tests. First one payload dot get

is the review body and the second one will be payload

Don't get

IMDb ID Okay. Finally we say

comma HTT be

status dot created, it will be 201. Since we are creating a new

review, we want to send 201 instead of the plane 200 Because

this means created and that means okay. So now that we have the post mapping done, we need our REST client a separate risk

flying to test our test out our code. But before that, I can see

that there is this squiggly line underneath the view controller. So I will just go to problems and see what's wrong. cannot

resolve symbol IMDb ID Okay, let's see where I missed a typo here. So it should be inside a pair of quotation mark. And

that's gone. See, this is the beauty of IntelliJ IDEA. It helps you a lot. Now let's restart our program. And I will

use postman to test out our API.

In fact, I just thought instead of mapping the reviews request

to API slash API slash movies, let's just go with reviews.

Because that sounds more appropriate. Now let's start our application

and let's jump to postman.

So inside postman first, I'll create a new collection. And I'll call the movies API.

Then let's create a new GET request get all movies

so it would be localhost 8080 slash API slash VLANs slash x.

So it works out just fine. Let's copy the IMDB ID from one of

these movies so that we can test out our next request

which will be get a single movie

and we should be able to carry everything from the previous request

will host slash view and slash movies slash then the IMDB ID.

Let's execute and the works just fine. Finally, let's try to create a new review here to review.

And let's say v1 slash reviews.

Okay, that's I set that because by mistake so it has to be post

inside body I would say raw data in JSON format.

So I will say review

body will be I really enjoyed the movie with

I. Yeah. And then I will say IMDb ID,

column and timed ID. Now you have to make sure that the name

or the key in this JSON data review body and IMDb ID

exactly matches the two keys inside the payload which is

review body and IMDb ID. So what will happen is this endpoint will receive a JSON data from the user and then convert it to

a map where the keys are extreme and the values are history. Then from this map, we'll be able to access the review body which is

a string and the IMDB ID, which is another string. And then through the service layer, we can create a new review on the

database, update the movie to be associated with that review, and

then return the review. And of course, there's a repository that works as the intermediary layer

between the service class and the database to let's let's go

let's come back postman, and let's see if it actually works on

here. So we have got 201 created. So theoretically, we should have created a new review for the movie boosting boards.

And let's go back to get a single movie it sent,

come down to review ideas. And of course, we have the new review associated with this movie.

That's how we can get the list of review ideas, they will be

embedded with the data of the movie they're referred to.

So in terms of the API, that's it, I know, it's a pretty simple API. But the goal of this course, or the first half of

this course, was to give you some idea about how you can create a new back end API. Powered by Spring Boot and

MongoDB. I hope that you have understood the basic concepts.

using MongoDB. And Spring Boot together, you can go ahead and

further your study by learning about how to implement authorization and authentication. You can also

learn about more complex relationships, and maybe try to build a bigger API with Gillette and ports and other kinds of

requests. I will be leaving this stretch right now. And from now

on Gavin long will teach you how you can create a React application that can go with this API. So take care, I will

maybe see you in another course in the future. Now that you've learned how to develop the back end, Gavin will teach you how to

develop the front end using React. Just a quick note before we get started, this will not be a deep dive into React. My only

goal here is to demonstrate how we can interface with a remote API from a front end application created using React through

which we are able to retrieve data from a MongoDB database hosted on a remote server, as well as posting data to our

MongoDB database. Now, any data that we retrieve from the server

will be in JSON format. JSON formatted data may be pleasing to some people. But most users of application I think would

much prefer something like this. So we're going to transform the data passed to us in a JSON format into an aesthetically

pleasing front end display. React and associated technologies provide an efficient way to create both

aesthetically pleasing front ends, as well as help us developers to facilitate a great UX user experience. I'd like to

give a shout out to the movie db.org All the movie poster images and backdrop images that we are going to use in the React

application that we are going to build in this course, come from the movie db.org Right, let's get into it. Let's start by

launching VS code.

And let's create a working folder on our local machines.

I'm going to name my folder movie client.

Let's open our working folder like this from within VS code.

So we are going to use the Create dash react dash app command to generate our React projects infrastructure. You of

course, must have Node js, preferably the latest version installed on your local machine as a prerequisite before

creating a React application. If you don't have no JS installed, you can navigate to

this URL to install the latest release of node j s. So from

within VS code, let's launch the terminal window so that we can create our React project infrastructure using the Create

dash react dash app command. You can launch the terminal window from within VS code by

pressing Ctrl and the tilde character.

Great. So the current directory reflected at the terminal prompt

should be the working directory that we have just created. my working directory is named movie client.

Then at the terminal prompt, let's type in MPX space create

dash react dash app followed by the name we'd like to call our

React project. So I'm going to name my react project movie dash

gold dash v one Then let's press the enter key. And the Create dash react dash

app command generates the React project infrastructure, which includes all the relevant node modules that contain components

that we are able to import into our React application. This process can take a while to complete.

And we can see the process on my local machine has now completed. So at this stage, we don't really need the movie client

folder. Opened within VS code, we only want the folder containing the files for our React project. So at the

terminal, let's use the cd command and then enter the name of the React application that we created using create dash react

dash app. So I'm going to type cd space movie dash gold dash v

one, and press the Enter key. So once we have made the folder containing the files for our React projects infrastructure,

we can launch another instance of VS code by typing and code space dot and then pressing the Enter key. This command opens

another instance of VS code, where we are now directly within the folder that contains the files for our React projects

infrastructure. We can now close down the first instance of VS code, because we

only need the instance that we have just launched. So we are almost ready to create our React application.

We won't be using some of the files that the Create dash react dash app commands generated for us. So let's delete the

following files from within the SRC directory. Let's delete the setup tests dot j s file, the report vitals dot j s file and

the app dot test dot j s file.

We must also delete a setting from within the package dot json file, which has to do with linting. We don't need this

linting setting in our project. Let's save our changes.

Lastly, let's open the index.js file and remove the code related to the report web vitals functionality like this.

Let's save our changes

and let's move on to installing a few NPM packages that contain components that we are going to integrate into our application.

If your project terminal is not available within VS code at the moment, you're able to launch the terminal by pressing Ctrl

and the tilde character. The next package we are going to install is Axios Axios will make

it easy for us to make HTTP requests from our React client

to the relevant API hosted on a distant remote machine. So that

we can retrieve relevant movie data from the MongoDB database also hosted on the relevant distant remote machine through

HTTP GET requests. We will also be able to add movie reviews to the relevant

Mongo DB database through the use of HTTP POST request, which

will be facilitated on our React client through the use of Axios.

We'll implement this functionality later in this video.

So to install Axios, we can type this command at the command prompt NPM space install space Axios.

And let's press the enter key.

Great, so let's install our next NPM package which will enable us to use Bootstrap for layout and styling

purposes from within our React project. So let's type the following at our command prompt

NPM space install space Bootstrap.

Note that in order to integrate bootstrap into our project, we must include this import statement within the index.js

file.

Let's save our changes. The next NPM package that we must install

allows us to use Bootstrap related components for layout and styling purposes. So let's install react dash Bootstrap.

For this purpose, we can install react dash bootstrap by typing

NPM space I space react dash Bootstrap.

And of course, follow this by pressing the Enter key.

Great. The next NPM package that we are going to install enables

us to easily install font awesome icons within our React

application. The first font awesome related package can be installed by typing in npm space I space at Fort dash awesome,

forward slash react dash Font Awesome. So make sure you type at Fort dash awesome here and not at

font stash awesome, which I know is a little bit confusing. So it's add 14 Awesome dash, react dash and then Font Awesome.

Great. Let's install the second font or some related package by typing

in npm space I space at Fort Awesome forward slash free dash

solid dash SVG dash icons.

And let's Of course, press the enter key.

Excellent. To play movie trailers from within our application, we're

going to use the React player component. So let's install the appropriate NPM package. So that we can integrate react player

into our React application. So let's type npm space I space

react dash player

and press the Enter key.

Great in order to declare the paths where our components will

reside, in order to map the routes to our components as it were, we are going to use relevant components from the

React dash router dash DOM NPM package. So in order to install

the React dash router, dash DOM NPM package, let's type in this command npm space I space react dash router dash DOM.

And of course, press the enter key.

Great. On our homepage, we want to display relevant movies in a

carousel. We can easily implement a carousel using material UI. So

in order to do this, we must install three material UI NPM packages. So we can install the relevant three NPM packages by

typing the following NPM space install space at Nui forward

slash material space at emotion forward slash react space at

emotion forward slash styles.

And let's press the enter key.

Excellent. In order to use the material UI carousel component, let's

install the material UI carousel NPM package. We can do this by

typing in npm space install space react dash material, dash

UI dash carousel

and press the Enter key.

Great, and we have now installed all the NPM packages we need in order to integrate the functionality we want to include

within our React application. Excellent. Let's get started with writing the code for our

React application. So the first thing I'd like to do is set up Axios. So to do this, let's create a folder within the src

folder

and Let's name our folder API. Let's create a file within the API folder named Axios config

dot j s.

We can import Axios into our project with this line of code.

So we can configure and export the Axios object that we will use to make HTTP requests to the relevant remote API through this

code.

Notes the settings that have been included here, we have the base URL setting, which provides the base address

of the API endpoints that our client react application will be calling. So you'll see later on when we use Axios. To call an

endpoint, we won't need to repeat the base URL with each

HTTP request within our code, we will only need to include the additional path information required to target a specific

endpoint. If this isn't clear, at the moment, it will become clear. When we write code to make a HTTP request to the

relevant remote endpoint. This setting here is necessary. Because during the development phase, the technology that the

remote machine is using to expose the relevant API endpoints is called ngrok.

I won't go into the specifics of N grok. But we need to include this setting in order for our client HTTP requests to not be

blocked by cause I won't go into the details, of course, but it stands for cross origin resource sharing. So all we need to know

is that because the relevant web API is running in a different domain, or origin, that cause may block our access to the end

point. The server code has included settings in order to

overcome the restrictions imposed by cores. And we are including this setting in the HTTP headers of our requests

from the client, so that we can overcome the restrictions imposed by cause IE, so that we are able to access the resources

made available through the relevant API endpoints. Great, so we have now set up Axios. Let's open the app.js file and

write code to call an endpoint that will return an array of movie data.

Let's write code to import our Axios object from within our Axios config.js file.

Let's import the use state hook and the use effect hook from react.

Let's return a D structured array from the use state hook.

The first item in the de structured array is named movies and will store an array of movie data returned from a call to the

relevant API endpoint. The second item in the array is a function that can be used to

change the state of the movies variable. When the state of the variable tracked by react through the use

state hook is changed the component is re rendered by react. So in this case, the app component will be re rendered

when the state of the movies variable changes. Let's create a function that will handle a HTTP get request to an endpoint that

returns an array of movie data. Once the movie data is successfully returned, this code that changes the state of the

movies array is executed.

You can see here we are passing the additional path information to the get method. This path information is appended to the

base URL setting that we created when setting up the configuration for Axios.

We are using async await for asynchronous thread management functionality to the client, I won't go into detail explaining async await

and promises. But this functionality can be effective in ensuring that the UI is not blocked. When potentially long

running operations, like for example, a remote API call are processed, the UI thread will not be blocked, and therefore

the use of screen will not, for example, freeze, the use the screen will still be responsive, while a potentially long running

IO bound operation is processing. Once the relevant task has completed, the code directly below the code that

kicks off the awaited process will be executed. So it's like a

promises made to return to the appropriate line of code. Once the task potentially long running IO bound task, for

example, has completed. As explained, this async await functionality is useful in order

to ensure a better UX user experience. In case something

goes wrong during the execution of our HTTP request, let's wrap the relevant code in a try catch block. And let's include code in

the catch block to log the relevant exception to the browser console window. So you could be more thorough here in

your code and check the HTTP status code. For example, if the HTTP status code returned from the server is 200. This

indicates that the operation performed on the server was successful.

I'm not going to include this functionality in this course, in order to keep the code fairly basic for our HTTP requests that

we are executing from the client.

Let's implement the use effect hook so that the get movies function is executed when the app component first loads.

Let's also write code to log the results returned from the call to the relevant endpoint to the browser console window.

Let's use the npm start command to run our React application.

Let's look at the browser console window to see if our results have been returned from the server.

Excellent our HTTP get request made through the use of Axios executed successfully.

So the next step is to start implementing our routing functionality or routing functionality.

Let's create a React component named layout dots Jas.

Let's open the layout.js file and import the outlet component

from the React Router DOM NPM package to generate boilerplate

code for our layout component, we can type ra F C E and press the Tab key like this.

Let's rather use semantic HTML here and include main tags instead of div tags. Within our main tags, let's include a

reference to the outlet component.

Let's go back to the app.js file and write code to import the layout component that we have just created.

Right, so I'm going to create a components folder within the src folder, and move the layout.js file into the components folder.

Let's write the code to import the layout component into the app component.

Let's write code to import the routes component and the route component from the React Router DOM NPM package

in the return part of the app component, ie where JSX code is returned from our component, let's write code to establish

the route mappings as it were for our applications React components. So let's first add the routes.

elements here

within the root element, let's include a parent root element. It's a parent root element because child root elements will

be included within this parent root element. Like, for example, a child route component that points to the home component we

will write the home component in a bit, will be a child of the parent route component that points to the layout component.

So the layout component referenced through the outlet element in the layout.js file denotes the components pointed

to by the child root component references that will include within the parent root element here, we'll include the relevant

child root elements as we progress with the development of our React application. The next step is to open the index.js

file and write code to import the browser router component,

the routes component and the route component into the index component.

Let's include code to establish the relevant route mapping for the app component.

Great, let's create our home component. Let's first create a folder within the components folder named home. Let's ensure

that the first letter in our home folder name is lowercase.

Let's add a file named home with an uppercase H to our home

folder. Let's type in ra f c e followed by pressing the tab key

to generate boilerplate code for the home component.

Let's simply type in the text welcome so that we can test our route functionality.

We'll create the code for our home component in just a bit

let's create the code for the route element that points to our home component within the app.js file.

Oops, I can see a little bug in our code. This reference to layout

is not correctly formed. Let's include the layout element properly within our code here.

Let's type in npm. Start at our command prompt and press Enter to launch the application

Great.

So let's create a component that will be a child component of the home component. We will name this component hero. This

components denotes the hero section of our home page that will display items in a carousel that are representative of

movies to the users of this application. So let's create a folder within the components folder named hero.

Within the hero folder, let's add a file named hero dot j s.

Let's also add a file to the hero folder named hero dot css where we will of course include CSS code for our hero component.

Let's write code to import the hero dot css file into our hero component.

Let's import the carousel component from the relevant man Terrio UI NPM package

let's import the paper component from the relevant material UI NPM package

let's write code to destructure. The props passed into this component, only one destructured value containing an array of

movie data needs to be included here

let's implement the carousel functionality. Let's include the carousel element like this.

Within the carousel element, let's write code to map each item in the movies array to an item displayed in the carousel.

The pay per element encapsulates each movie item displayed in the carousel.

Let's include div tags, so that the movie poster is displayed, as well as the movie title is displayed for each carousel

item.

As you can see, I've included CSS class references in the relevant div tags, we'll style our components in the hero dot

css file in a bit, where we will implement the appropriate CSS properties for

these CSS classes. Oops, let's make sure that we are importing the hero dot css

class correctly here. Let's open the home.js file and include the hero element in the

part of the code that is returned as JSX code from the home component.

Let's make sure that we are importing the hero component into our home component.

Let's write code to ensure that the movies array is being passed down from the app component to the home component and

subsequently the hero component.

And that's great. We are displaying the relevant data from the JSON data returned from the relevant API's endpoint. But

let's customize the styling of our carousel through CSS code.

We can position our Vyas code editors to a pane on the left side of our screens. And we can position our browsers to a pane

on the right side of our screens like this. In this way, we can watch the effects of our CSS code changes in real time. I'm

not going to go into a lot of detail regarding the CSS code. The code will be available on GitHub, where you can examine it

in more detail if you'd like. Let's first update the CSS code in the app dot css file. This

CSS code effects all components that are child components of the app component.

And you're able to observe the effects of the CSS code on the browser in real time as the CSS code evolves.

So let's import quicksand font from Google with this line of code

let's set up the box sizing and font family properties for all elements

let's set the body elements padding and margin properties to zero.

Let's style the App class. Note that all of our components are children of the app component.

Let's remove this code from the index dot css file because it is not needed

let's style the carousel component that has been implemented within our hero component. So let's include our

CSS code in the hero dot css file.

And we are able to watch the effects on the carousel displayed in our browsers in real time as the relevant CSS

code evolves.

And we don't actually want the post image to take up the carousel display. As it is at the moment, we want an

appropriate backdrop image a larger image to fill the background for each item in the carousel.

I want to include a gradient that fades from dark to light from the bottom of each carousel item to where the title is

displayed. In order to dynamically reference the background image URL for each carousel card,

I'm going to use a CSS custom variable. So we can do this by including an inline style within the div

element that references the movie dash card CSS class. And within that inline style, assign a variable to the property that

we wish to reference within the relevant CSS file. So here, we

are naming our custom CSS variable image.

We are setting the CSS URL value to the CSS background dash image CSS property from the backdrops property of each movie in the

movies array returned from the server. Note that the backdrops property returned from the server is an array containing

pause to movie backdrop images. In this code, I'm choosing to

reference the first backdrop image path in the backdrops array returned from the server. Each backdrop points to an image

that we want displayed as a background image appropriately for each movie item in the carousel. We can now reference

each background image from within our CSS class and style the background appropriately.

So we are assigning the background dash image CSS property or gradient that fades from light to dark?

This is just a stylistic choice. For our front end display, you can see here in our CSS code how

we are using the dash dash image custom CSS variable to link the

URL property value referenced in the JSX code in the hero component to the background dash image CSS property in the hero

dot css file. You can see that the poster image is looking distorted so let's style the poster and the title

appropriately.

Great and we have used absolute positioning and the CSS Flexbox

to style our poster and handle where it appears in the layout

let's style the title appropriately.

Great, and our carousel is looking pretty good. Let's create the components to house our navigation display. Let's

create the header component. So let's create a folder within the components folder named header. Let's create a file within the

header folder named header.js.

Let's import the appropriate font awesome components

let's import the appropriate bootstrap components.

And let's use React bootstrap components to create a responsive layout for our navigation menu which will be

displayed at the top of the screen. This is one of the huge advantages of using Bootstrap it can save us a lot of time in

creating responsive layouts.

We are using the FA video flash font awesome icon for our logo that will be displayed in the top left corner of our screen

within the navigation bar that we are creating within the header component.

The login and registration buttons are being included here. Superficially, they are purely cosmetic indicating that we

could extend the functionality of this application to include login and registration functionality at a later stage.

For example, if we wanted to enable users to create their own list of favorite movies or a watch list, the watch list link

here is also there purely for cosmetic reasons. We won't implement a watch list component in this course.

So above the routes functionality within the app.js file, let's reference the header component

let's go back to the header.js file and make sure that our imports are correct.

Let's run the code

Excellent, our navigation bar, and our logo looks pretty good

the next step is we want to be able for a user to click a play

button icon on each carousel item. And for an appropriate movie trailer to play will enlist the help of the React

player component for this purpose. Let's first create a folder named trailer within the components folder.

Let's create a file named trailer dot j s within the trailer folder. Let's also create a file named trailer dot

css within the trailer folder.

Let's create the relevant input code in the trailer.js file.

Let's import the use params hook into our code from the React Router DOM NPM package, a YouTube video ID will be passed

into this component as a parameter. So we are going to use the use params hook to extract the relevant parameter

value from the relevant URL. The relevant parameter will contain

a YouTube video ID which will allow us to play a YouTube video, which is a trailer of the relevant movie to the user.

Let's import the React player component.

Let's import the trailer dot css file.

Let's create the boilerplate code for our trailer component by typing in our AFC E and pressing the tab key. Let's

create a div container in which our React player components will reside

let's assign the Paston parameter value to a const named key

then we can include the React player component appropriately like this

we want the player to include controls. So here we can set the controls property to true we can set the playing property to true

so that the relevant video plays as soon as the trailer component loads

and lastly, and most importantly, we can set the URL property to the relevant video on YouTube so we can append the

relevant YouTube video ID stored in the key const to the base YouTube address to point react player to the YouTube video that

will be played within react player great

let's go to the app.js file and include the relevant route information for the trailer component

notice how we are able to let React Router DOM know as it were about the YouTube video ID parameter then our code within

the trailer component must access

let's include a play button icon provided by font awesome within our carousel.

Okay so the play button icon is present, but it needs to be appropriately style Okay, as it is tiny at the moment, let's go to the hero dot

css file and style the play button icon appropriately.

Great, let's include a media query so that our carousel adapts to smaller screens

Great, let's go to the trailer dot css file and create CSS code

so that the React player takes up 90% of the viewport height.

Lastly, we need to create the link so when the play button icon is clicked, that the trailer component is invoked,

and the appropriate parameters passed through to the trailer component.

So the trailer link property retrieved from the server contains the entire URL to the YouTube video, we only want to

pass the ID of the YouTube video to the trailer component from the hero component. We can extract the YouTube video ID by

using the JavaScript substring method to extract the last 11

characters from the trailer link property like this

let's run the code.

Excellent. The last component we need to create is the reviews component. The reviews component will enable users to view

reviews from other users as well as add reviews for movies into the system, ie, which entails a HTTP POST request being made.

From our React component to the relevant remote API endpoints carrying the review data entered by the user and the server side

Java code will then save the review data appropriately to the MongoDB database. Firstly, let's create a component named review

form. This component will contain a form component that contains a text area control where the user can enter text

that denotes a movie review. a submit button will also be included so that the user can click this button in order to

submit the user's movie review. So let's create a folder named review form.

Let's create a file named review form dot j s within the review form folder.

Let's include the code for the functionality we have just discussed.

Let's create the reviews component that will be a parent component of the review form child component. Let's create a

folder named reviews. Let's include a file within the reviews folder named reviews dot

j s

let's include the appropriate import code at the top of the reviews.js file.

Let's generate the appropriate boilerplate code for the reviews React component by typing in RFC E and then pressing the tab key.

Let's use the use ref hook to reference the text area control within the review form. Let's use the use params hook so that

we can extract the movie ID parameter value from the relevant URL.

This movie ID is actually the IMDB ID of the movie that we

have retrieved through the API call that is made to retrieve the array of movie data from the server. The IMDb ID value, of

course, is used to uniquely identify a specific movie.

So when our component first loads, we want to call a method that is passed in as a prop to our

component in order to retrieve the appropriate data for the relevant movie that the user wishes to review. So we can use

the use effect hook like this for this purpose.

Let's use React bootstrap components for example, the row and col components to create an appropriate layout for the

reviews component, we want to display the poster for the relevant movie in the left part of the screen and the reviews in

the right part of the screen we can use the row and col components appropriately to achieve this.

Let's include the review form element here like this.

Let's create a method named add review and create the relevant HTTP POST request functionality for adding a review to the

MongoDB database which resides on a remote server.

And let's write code to update the state of the reviews array on the client side. optimistically meaning that we

are not using data returned from the server ie data that is saved to the database for updating the state of the reviews array. We

are updating the array directly from the data entered on the client.

We also want to include code that clears the relevant text area control. Once the user has successfully submitted a review,

we are updating the state of the reviews array on the client through the use of the set reviews method that will be

passed as a prop from the app component.

Let's go to the app.js file and create a method that uses Axios

to make a get request for data pertaining to a single movie

so an IMDB ID value will be passed into this method to the movie ID parameter and HTTP get request. Using the movie ID

value appropriately in the URL will be made to retrieve data for a single movie from the remote server.

So let's set the state for the movie variable.

And to make our code clearer, let's extract the reviews array from the movie data and track the state of the reviews array.

Let's make sure that the appropriate props are being passed down to the reviews component.

We are now able to create reviews for the movies retrieved from the remote server ultimately from a MongoDB

database that stores our movie data. So we are adding reviews to our MongoDB database through an appropriate HTTP POST request

to inappropriate endpoint Excellent.

One thing we must do whenever we implement a list of uniquely identifiable AI items in our JSX code like this

is appropriately includes the key property for each list item. to uniquely identify each item in the carousel, we can set the

key property for each movie item displayed within the carousel to

the appropriate IMDb ID value for each movie. This value has

been passed down to us with the movie data array that we retrieved from the remote server using the appropriate HTTP get

request. I hope you've enjoyed building this front end using React. And I hope you feel that you have benefited from this

course. If you've got to the end of this course you've done exceptionally well. Thank you and take care