earn how to build a complete full stack movie streaming app with AI powered movie recommendations you'll use Go with the Jin Ganic framework on the back end React on the front end and MongoDB for data storage for the AI features you'll connect your Go backend to OpenAI using Langchain Go along the way you'll set up authentication with secure token handling making sure access tokens are stored in HTTPON cookies to protect against cross-sight scripting attacks you'll also learn best practices for securing client server communication over HTTPS once the app is built locally you'll deploy it to the cloud with MongoDB Atlas render for the Go API and Verscell for the React client by the end you'll gain experience not just in Go MongoDB and React but also in cyber security AI integration and cloud deployment gavin lawn developed this course and MongoDB provided a grant to make this course possible hi everyone and welcome i'm Gavin Lawn i'm super excited to bring you this course where we'll build a full stack web application stepby step using cuttingedge technologies we are developing this application for a fictional company named Magic Stream that provides Why'd you do it i didn't a movie streaming service as well as an AI powered personalized movie recommendation service that leverages a technology called lang chain go for interfacing from our go or golang code with open AI we are going to use React on the front end to leverage high performance front-end user interaction functionality so as to provide our users with a superb UX user experience we'll create the serverside code for a web API component using Go also known as Golang go or Golang is a statically typed compiled programming language created by Google it was designed with simplicity efficiency and scalability in mind making it well suited for modern systems programming cloudnative applications and large-scale distributed systems i have to say after working with Go lately it has become one of my favorite programming languages you'll see it's just great to work with it's a great modern simple yet powerful language to learn and it also pays really well for our data storage facility we'll use the awesome MongoDB a NoSQL document-oriented database designed to store query and manage large volumes of data in a flexible and scalable way mongodb is optimized for fast read and write operations especially useful for real-time applications as mentioned earlier for creating a super responsive front end we'll use React in order to provide our users with an excellent UX user experience react is a JavaScript library not a full framework used for building user interfaces especially single page applications SPARS where the UI needs to update efficiently in response to user actions or data changes it was originally developed by Facebook and is now open source and widely adopted so this course has a bit of everything learn a modern cuttingedge programming language like Go create a full stack distributed web application learn important aspects of cyber security learn how to integrate AI into your applications and then at the end of the course when we deploy our app to the cloud we engage in aspects of DevOps once we've developed our application we'll deploy the serverside web API component written in Go to Render a modern cloud hosting platform that provides developers with a way to easily deploy and scale web applications we'll deploy the React client code to Versel a cloud platform optimized for front-end development and we'll deploy our MongoDB database to Atlas atlas is MongoDB's fully managed cloudnative database service sometimes described as database as a service so this web application is a great example of the implementation of a loosely coupled architecture where functional entities on the web the serverside component written in Go and the client written using React interact via HTTP as one full stack web application we'll also include cyber security features like token authentication over HTTPS to protect our data so as mentioned earlier I'm Gavin Lon i've been developing software professionally for decades at this point and love sharing my knowledge i run my own YouTube channel where I teach programming and discuss technology in general i'm proud to say that I've created several courses for free code camp most of them dedicated to teaching programming but I must confess that I went to the dark side last year and taught a course on video editing using Da Vinci Resolve so please check out that course if you want to add video editing to your skill set okay so enough about me let's discuss what this course is all about i've created a fictional company called Magic Stream we are going to develop a movie streaming service for Magic Stream in the form of a full stack web application that provides a special feature that uses AI to recommend the top five movies tailored to the user's specific tastes this recommendation service is based on the users's movie genre preferences and the sentiment derived from movie reviews created by the administrators of our system so the workflow here is an administrator logs onto the system and creates a movie review in natural language for a movie offered for streaming through magic stream when the administrator submits the movie review a custom prompt that we'll create is submitted along with the movie review to OpenAI via a technology called Langchain Go the relevant LLM large language model returns a response in one word describing the sentiment behind the movie review our prompt includes instructions to only return one word describing the sentiment behind the movie review for example the sentiment could be excellent good okay bad or terrible this sentiment described in one word is then saved to our database along with a unique numeric value associated with each sentiment you can see how it is easy for our system to include a value for each of these sentiments so for example excellent can have a value of one good two okay three and so on the last one terrible has an associated value of five we can then query our MongoDB database for movies and order them by their values that denote the sentiment for each of the movies stored in our database it is important to note that these sentiments are extensible we are going to architect the application so that we can extend the number of possible sentiments associated with movie reviews we could later include a wide spectrum of sentiments and associated values for example you could include the word unwatchable below terrible where terrible has an associated value of five and unwatchable has an associated value of six so these values can be used as a ranking order for the relevant recommended movies this course has pretty much got everything i mean we are also going to include cyber security features like tokenbased authentication over HTTPS i'll firstly show you how the access token is generated once the user authenticates with the system i.e by logging on with the user's credentials then for subsequent HTTP requests the token is passed through the header of the relevant HTTP messages so that the user can be authenticated before accessing protected endpoints but it doesn't stop there by passing the token through the header our application is still vulnerable to XSS attacks cross-sight scripting attacks where malicious JavaScript code can potentially be injected into the client side and read the access token an access token is like a key that can be used to unlock protected resources so we must protect the key how can we do that i'll show you how you can avoid storing the tokens on the client side and passing it through the header by instead storing the access token as HTTP only cookies which means the tokens cannot be read through JavaScript code and therefore mitigates the risk of the access token being stolen i'll show you how to do that in detail in this course so these are very important cyber security related practices to learn to leverage Go for our web API component we are going to use a web framework called Genonic genonic sometimes referred to as Jyn is a high performance HTTP web framework for go Golang gingonic is lightweight and fast minimalistic but powerful jinggonic i know it sounds a bit like jin and tonic we're not going to be vibe coding in this course oh sorry whiskers i feel it's important to learn to code at least at the beginning without the use of AI assistance so it gives you a tactile and deep sense of the development process and building up the code step by step without AI assistance in my view is the best way to understand programming and associated technologies so we are going to build this full stack web application step by step and we are going to have a great time doing it so as I said at the beginning this course has a bit of everything learn a modern cutting edge programming language like Go create a full stack distributed web application learn important aspects of cyber security learn how to integrate AI into your applications and then at the end of the course when we deploy our app to the cloud we dabble with DevOps dabble with DevOps we engage with aspects of DevOps so we are going to have a great time developing and deploying our full stack web application right let's start from the beginning which involves setting up our local development machines with the technologies we'll use to develop and test the application let's get into it right so to install MongoDB we need to navigate to this URL so let's go through Chrome i'm going to use Chrome as my browser and I'm going to navigate to this URL www.mongodb.com/try/d download/ community then press the enter key and you can see the website has appropriately detected my platform settings so we have the platform setting for Windows x64 which is the platform I'm using uh which I'll be using to develop the application for this course you have a choice between MSI and Zip here but we want the MSI package and this is the appropriate platform for us but we have a list of various other platforms we have Mac platform and we have various distributions of Linux available to us here but as stated we want the Windows x64 platform selected here so the next thing we do is just hit the download button to download the relevant MSI file once it's downloaded we'll double click on the relevant MSI file and run the relevant installation package you can see we've got an indication as to how much longer it will take to download the full MSI package so we just need to be a little bit patient here 2 minutes left it says not too long let's fast forward and please note that this is the community edition of MongoDB server and it's absolutely free to use on our local machines all right not long now 9 seconds 8 seconds 7 seconds 6 5 4 3 2 1 okay great and now we should have the folder icon displayed to us here it is and we can just navigate to our downloads folder using that there so to install MongoDB on our local machines we simply doubleclick this file here which has themsi extension excellent the setup wizard will install MongoDB 8.0.11 blah blah blah blah blah click next button excellent and then let's accept the licensing agreement by selecting this checkbox here and then let's press the next button and we are recommended to do a complete install here so let's hit the complete button and on this page I think we can just leave everything as is so let's press the next button to continue now this is a very important checkbox here let's keep this checked because this will install automatically a guey interface for MongoDB so we'll be able to use this guey interface to manipulate our data create databases collections documents within collections etc and it's a really great graphical user interface that we can utilize great so let's press the next button here and then the last step is to press the install button i'm not going to press the install button because I've already installed uh MongoDB on my local machine so I'm just going to cancel out of this but please go ahead and press the install button and go through with the installation process so the next step is we want to verify that we have indeed installed MongoDB successfully on our local machine so to do that we can do it through the command prompt like this so mongo d- version like this excellent so we've got version 8.0.8 which is the target version that we installed on our Windows platform excellent then in order to be able to interact with MongoDB with the MongoDB database on your local machines you can run MongoDB commands like for example query commands from your command line using the MongoDB shell this can be handy for test purposes please note that the installation of the MongoDB shell requires a separate installation to the one for installing MongoDB i.e the installation we have just completed and verified so to install MongoDB shell it's www.mongodb.com slashtry slashd download and then the word shell slshell like this press the enter key and you can see now that the actual website itself has detected my particular platform Windows x64 so I've got Windows x64 version 10 plus so I'm actually on version 11 of the Windows 6 x64 operating system which is correct and then here we can download the relevant MSI file and then we can just run that MSI file from our downloads folder by double clicking on the relevant file and going through the relevant installation instructions then to verify that MongoDB shell has actually been installed and I have actually gone through the installation process so I've got MongoDB shell installed you can verify this by typing MongoDB so sorry it's not MongoDB it's Mongsh version like this 2.5.0 so I've got the appropriate version of Mongo MongoDB shell installed on this local machine and we verified it with this command excellent uh sometimes it won't automatically configure your path environment variable which can lead to problems so to verify that our path environment variable has indeed been configured correctly we can go to this utility cis dm dot cpl press the enter key then we can go to this tab the advanced tab and press the environment variables button here and let's look inside the system variables list here and navigate to the path environment variable hit the edit button and let's see if we can find the MongoDB environment variable that we want sorry well the path environment variable and make sure that MongoDB is appropriately set up within the path environment variables and here it is c program files MongoDB server 8.0 bin and if we copy that to our clipboards and we press Windows key R here and use Windows Explorer we can navigate to that path and we can see here that the relevant files have been appropriately put into that directory structure and we are appropriately pointing to that directory from within the path environment variable so that's excellent great and then the other verification we should do is check that MongoDB is actually running in the background and we can do that by running our services so Windows keyr type services.msc like this and then type select the list and type mo and there we are we can see our MongoDB server is running in the background here and we can use this facility to stop or start start our server but we don't want to stop our server we want to keep MongoDB server running in the background the next thing we want to do is launch Compass so we can see the graphical user interface that ships with this installation of MongoDB and then let's press on the connect button here next to main connection here excellent as you can see I've been playing around with a number of MongoDB databases and this is essentially the structure we're about to create so we're going to create a new database next to the main connection node here let's press the plus button and then let's give our database a sensible name so let's call this magic stream movies like this and our first collection will be named movies so let's enter it here collection name and then we just press the create database button to create our database excellent so I want to create three other collections within this database so let's hit this plus icon here to create a new collection and let's name this collection users this will store information regarding the users of the system i'm going to import three data for three users initially so that we can uh test our application as we develop it using that data initially and then we'll also create a registration facility where we can add further users but we'll get to importing the relevant seed data in just a bit so let's create a new collection let's call this one genres like this let's hit the create collection button here excellent and we've got one last collection that we need to create so let's hit the plus button here and this collection's name is rankings and we'll see the data for the rankings collection in just a bit but let's first import the data for movies now to import the seed data you can firstly download the relevant data from my GitHub repository so let's go to my GitHub repository here and if we go to Magic Stream and this folder Magic Stream seed data you can see a number of JSON files here that I recommend you download to your local machine and then you can import the relevant data into your collection through a utility within Compass and I'll show you how to do that right now so if we go back to Compass let's import the data for our movies collection and to do that we hit the add data button here import JSON or CSV file let's select that option and then doubleclick movies.json here and then hit the import button and you can see I've prepared offcreen 15 data for 15 movies and we've just imported them into our movies collection here you got Jack Reacher Blitz Star Trek The Undiscovered Country Star Wars Empire Strikes Back what a classic and we've got a few classic movies here the Good the Bad and the Ugly classic from the ' 60s unforgiven awesome movie great and then I was talking about the rankings now we're going to create a facility that interacts with AI with OpenAI and it's going to extract sentiment from movie reviews written in natural language you can see here within each document each movie has a movie review um so for example this movie is awful the AI will extract the sentiment just in one word terrible and the ranking value makes that sentiment quantifiable so we're taking a movie review written in natural language and we're quantifying its sentiment and you'll see how useful that can be later on when we develop that part of the application relevant to the AI functionality excellent and so let's import the data for the rankings collection and it's the same procedure we go import JSON like this and double click on the rankings.json file and hit the import button here and there we have it and you can see the AI will select from the admin review written in natural language one of these options excellent good okay bad or terrible and in doing so the AI is taking qualitative data written in natural language and turning it essentially into quant a quantifiable value which is represented by each of these sentiments so we've got excellent good okay bad and terrible to choose from here right let's import the genres and it's the same procedure you click this dropown go import JSON or CSV file and then double click on genres.json and click the import button here excellent and lastly we have the users let's import the data for the users collection like this let's hit the import button and we've now imported three users one of which is an administrator so Bob Jones is the only one because he's part of the admin role he's the only one who can actually add reviews to movies okay so the next thing I'd like to demonstrate is how we can use the command prompt to interact with our MongoDB database before we test the MongoDB shell I just want to clarify that this is just one way of interacting with MongoDB in fact in this course we are going to use a specific MongoDB for Go driver from our Go or Golang code in order to interact with our MongoDB database using the MongoDB shell can be a convenient way of interacting with our MongoDB database for example during an appropriate testing phase of your application or perhaps to diagnose potential database related issues during the development phase of your applications as stated in this course we are going to interact mostly with our MongoDB database in code via the MongoDB for Go driver we'll look at this in detail a little bit later so let's fire up the command prompt like this and then we need to go into the MongoDB mode if you like and we can do that with this command mongod press the enter key and as you can see we are automatically in a database named test here so it actually created this database by default when we went through the installation process so let's create let's run a command that creates a collection within the test database and we can do that with this command db.create create collection and let's name our collection customers like this and press the enter key okay one so that looks good it should have created our customers collection and we can verify that by going into compass here let's go into our test database customers and there it is there's our collection and let's create let's first clear the screen and let's create a document within our customers collection and we can do that with this command dbc customers doinsert one and then we can include the relevant JSON object like this and we're going to add a customer named Bob Jones so let's give this field the name name like this and then let's include the field value like this within quotation marks bob Jones and the email address email is Bob Joneshotmail.com let's press the enter key excellent so we've added a document to our customer's collection the document is denoted by the customer Bob Jones and let's verify that Bob Jones has been added to the customer's collection and there's the document for Bob Jones and you can see it's added a unique identifier for this particular document and let's for good measure let's add another customer to our customers collection this time let's add a customer called Sally James let's give Sally a sensible email address hotmail whoops should be at atotmail.com and let's press the enter key excellent and let's verify that Sally James has been added to the customer's collection and there it is Sally James brilliant and then if we want to display if we want to query for all of the documents within our customers collection we can run this command db.customers customers.find open close brackets press the enter key and there we have the two documents that we've just added to the customers collection displayed to us brilliant let's clear the screen um and let's actually go into the the uh and let's actually go into our magic stream movies database so to do that we type use magic stream movies like this so we've switched to Magic Stream movies and then we want to return all the movies that are saved to our movies collection so to do that we type db dot moviesov dotfind open and close brackets press the enter key and there we have our 15 movies brilliant excellent and let's say we wanted to just return one of those movies let's clear the screen first we can do that with this command db dot movies oops movies dotfind one and then each movie has a field named IMDb ID so you probably know this but on the internet each movie has a unique identifier it's IMDb number so I'm using that within our database to identify them to uniquely identify our movies so let's say we wanted to find the movie called Once Upon a Time in Hollywood so we can do that through its IMDb number or IMDb ID and to do that in code we pass in a JSON object to the find one method like this so this field is called imb_id so we don't need the quotations here but the field is called IMDb_ID colon and then we want to include the value for the IMDb ID field tt 713 6 22 and then let's press the enter key excellent once upon a time in Hollywood the document for Once Upon a Time in Hollywood is displayed to us and our command has executed successfully great right so to install Go on our local machines let's navigate to this URL so it's https colon slash slashgo.dev it's an easy URL to remember excellent and then let's press the get started button here and then let's press the download button here and you can see here we have a number of installations to choose from and it all depends on what platform you are running i'm running a Windows platform so Windows 10 or later this will be compatible with Windows 10 or later i'm on Windows 11 so I'm going to click on this link here to download the relevant MSI package and then we've of course been through this before we got our indicator here and it's downloading that pretty quickly so we don't have to fast forward anything and we're on we've only got say a few seconds left to go that's looking good oh a bit of a hiccup right let's fast forward and here we go a few more seconds six four three two no it's another Okay 4 seconds 6 seconds 9 seconds 8 seconds guess my internet's not particularly good today but we are getting there nearly there excellent done so now all we need to do is click on the folder icon here to go to our downloads folder and here's our MSI package and then to install go simply double click on the MSI package and follow the instructions provided to us in the wizard please wait while setup wizard prepares to guide you blah blah blah so the next button isn't yet enabled so we can't click on that just yet so let's just be patient here migrating feature states from related applications it's doing things behind the scenes welcome to the Go programming language AMD 64 Go 1.24.4 setup wizard the setup wizard will install Go programming language AMD 64 Go 1.24.4 on your computer click next to continue or cancel to exit the setup wizard let's press next and install Go so what's happening there is it's detected that I've already got Go on my machine so I'm actually going to exit but please go through the instructions go through the installation process to install Go right so I'm just going to go no exit here and then finish and just going to minimize this right and then just to verify that I've got Go installed successfully on my local machine I'm going to go into the command line here and type Go version excellent so 1.24.2 Windows AMD 64 that's excellent so I've got Go successfully installed on my local machine um just like we did when we installed MongoDB we should verify that Go is appropriately configured within our path environment variable so to do that we do the same procedure we type CIS DM CPL here and we go to advanced and to get to our environment variables we hit the environment variables button here so within our system variables let's find the path environment variable settings here press edit and let's see if we can find go within the paths configured here and here we go program files go.bin and we can actually copy that to our clipboards go Windows keyr and navigate to that directory and we can see we got go.exe exe there go FMT.exe so we've got the relevant executables in our Go directory and that has been successfully configured with for within our path environment variable so that's great okay so let's go out of this here and the next thing we can do is let's go to the command line i'm actually going to navigate to the C drive because we have configured go correctly within our path environment variable we should be able to run our Go commands from any directory so we're in the C drive here and let's create our first Go application so to do that I'm first going to create a folder so I'm just going to go mk dur and create a folder called go first app like this press the enter key and then I'm going to navigate into that folder that we've just created go first app excellent and then I'm going to launch notepad because for this first application we're just going to use Notepad it's going to be a very simple traditional Hello World application so we don't need a particularly sophisticated editor for that but when we get down to creating the web API that we're going to create with Go and Jenonic the Genonic web framework we'll use Visual Studio code for that but for now let's just use Notepad and create a basic hello world application so let's initialize our Go application with this command go mod init my app like that press the enter key and then let's launch notepad so we can just type notepad if you've got notepad appropriately it's set up you should be able to just type notepad to launch notepad and let's save this file as main.go go it's going to go to my C directory go first app there it is and then I'm going to create a file called main.go here set that to all files so that it doesn't mess with our extension here so main.go is going to be where we write our code and let's save that like that so firstly we want to designate this go file as part of a package called main so to do that we type package main like that press the enter key then we want to import a particular package that we can use to write out text to the screen and we can do that with this package so I'm just going to type import and it's the package we need is called fmt let's press the enter key here so every go application has a main entry point a method called main and that's the entry point of the application so let's create the function for this using go so we type funk and then the name main like this open close brackets and then we can open and close curly brackets and this demarcates where our go code will be where we're going to write some text to the screen functionality that writes text to screen so we're going to use that fmt package functionality and then the method is print ln and then we're going to write uh whoops hello go world like this let's give it an exclamation mark here let's save that and that's our code that's our first Go application written and let's see if we can run it right so I've cleared the screen and let's see if we can compile our Go application so to do that we go we type go build like this great and let's look in that directory and see if it's created an executable for us and it's created the executable called my app.exe so to run that we can just type my app like that hello Go world and we've written our first Go application brilliant right so as mentioned earlier we're going to be using on both the client and service side we're going to be using Visual Studio Code to develop our application so we of course want to firstly make sure that we have Visual Studio Code installed on our machine so let's type in https/ slash slashcode.visisualstudio.com/d download so that we can download Visual Studio if we don't already have Visual Studio on our machines and then it's just a case of going through the install instructions as we've done before so you want to download the installer by clicking if you're on a for example on a Windows 10 or 11 platform you want to download this installer here you want to click this option here if you're on a Mac platform you want to click this option here and of course this is for Linux so let's click on the Windows option here and we're downloading it excellent going very well here we've got our indicator and once it's downloaded of course all we do need to do is uh double click on the relevant file and go through the installation instructions provided to us through a wizard and that's it we've got the folder icon presented to us and then we've got an EXE file here which is our installer executable and we can just double click on this and install i accept the agreement next i would actually encourage you to select this check box here to click on this checkbox here so that the icon is readily available on your desktop once the installation process completes then let's press the next button here and then you press the install i've already got Visual Studio installed so I'm not going to go through this process but please go through and install Visual Studio Code if you haven't yet installed Visual Studio Code so the next thing I want to do is create a folder on my local machine so cdev and Golang and you can see I've done a number of incarnations of the magic stream application in preparation for this course but we're going to create a fresh new folder where we'll create our application and I'm going to call this folder magic stream movies like that press the enter key excellent so we've got our designated folder that will house the files for our application so we're going to now fire up Visual Studio Code and firstly create the infrastructure for our projects so we got a React project on the front end the client and we've got a Go which leverages Jenonic which is a web framework on the server great so let's open the folder we've just created the fresh new folder so open folder and we want magic movie no we don't want that one we want magic stream movies and let's select that folder if you already had Visual Studio Code installed on your machine you might not necessarily have the latest version and you can check for updates with through this menu option here check for updates okay there are currently no updates available so I've got the latest version so I'm good to go but if you haven't yet got the latest version just hit the check for updates menu option here excellent in this part of the course we're going to focus on the server side code but just bear in mind we're going to then once we finish the server side code we're going to come back and develop the client React code excellent so let's create a folder for the client React code and we'll call this folder aptly client and we'll get back to this a little bit later so let's progress to the server side code so let's create a folder called server which will house our server go code so let's create a folder called server and then within this folder let's create a subfolder called magic stream movies server like that okay excellent please note that if you get stuck at any point or you just want to reference the final code for the application please check out the final code at this URL there may be a few minor differences between for example naming conventions for the files when comparing the final code to what we've discussed in the video but the differences are very minor so please don't be put off by these minor differences okay so now let's invoke the terminal window you can do that by control back tick like this by default uh it has selected the root directory of our solution which will contain both client and serverside code so we're in the magic stream movies folder here and we're going to be using PowerShell here okay so within this directory here we want to create a module which will store all the references to the dependencies that we will be using in our Go project so to do that we first need to be in the root folder of our serverside code which is magic stream movies server and then we can type the command to create our module our go.mod file so to do that we type go mod in it so we're initializing our project and this uh file that we're creating will store all the the references to the dependencies that we will be leveraging for our for the server side part of our application which will be written in go okay so go mod init and then the name of our module this is a best practice will actually be where our serverside code will be stored on GitHub so we'll be uploading this serverside code to a designated location on GitHub and this is where our serverside code will reside when we eventually push it to GitHub so github.com/gavanlon digital so this is my path so obviously you need to upload it to your particular GitHub path so you you'll want to include where your code will be stored on GitHub so github.com/gavlon digital and then magic stream movies and then server and then magic stream movies server like that right so Gavinon so github.com gavlon digital magic stream movies server magic stream movies server that's where my serverside code will reside and as I said this is just a best practice what this will mean is that for example other users will be able to import your packages by using the go get command and then typing the relevant GitHub path to get your particular packages to import those packages into their applications so this is why it's a best practice it futureproofs your application great so obviously I haven't yet uploaded anything to this path but I will be uploading the relevant code to this path at the appropriate time great so I'm going to press the enter key here and it's created now the good file here right so before we start writing a little bit of code I suggest that we firstly install a very handy extension that will enhance our go development experience basically so uh to do that we go to the extensions tab here and let's search for Go like that and this is the extension we want to install and I've actually already installed it that's why this button text is set to uninstall so if I press it it will uninstall it but yours might be set to install meaning you haven't installed it yet and if it's set to install I recommend pressing the install button and installing this very handy extension here so this go extension contains the following handy functionality intellisense results appear for symbols as you type code navigation jump to or peak at a symbols declaration code editing support for saved snippets formatting and code organization and automatic organization of imports diagnostics build vet and lint errors shown as you type or on save enhanced support for testing and debugging so it's well worth installing this Go extension great so the next step is to create a file within the magic stream movies server folder called main.go so let's do that main.go as discussed earlier when we created our hello world application each go application has its entry point and the entry point of the application is denoted by a function called main so firstly let's name the package for the code we're about to write let's name it main like this excellent let's import fmt this is the package we are going to use to write out our text to the screen within the main method and let's now create the function main like this and let's use the fmt package print line and let's print out hello go world like that okay that looks good let's make sure we save our code right and now we want to run our code so to do that we can make sure firstly make sure you've saved the code within the main.go file and then we can run this command so it's just go run dot and that should run the code within our main function here but firstly make sure that you are in the appropriate directory the root directory of your serverside code which in my case is magic stream movies server like this and then go run dot should run our code excellent hello Go world great so now that our infrastructure is set up appropriately we're ready to write our web API code okay so firstly we want to make sure that we're in the same directory as the go.mod file so let's navigate to the appropriate directory so we want to be in server slash magic magic stream movies server like this and then we want to type the command that will import and install the ginonic web framework package so to do that let's type go get dash u like this and then we want the path on github to where the relevant package is so github.com jinonic slashjin like that whoops jin and then press the enter key and it's as simple as that and as always we have to be a little bit patient while the import and installation process is underway and we're finished now that's brilliant that wasn't too painful let's clear the screen and we're ready to develop basic functionality using the Genonic web framework excellent so firstly let's replace this code within the main method well let's get rid of this code and then we can start coding the main method which of course as discussed earlier is the entry point for our application but before we start writing the code let's have a quick look at go.mod here and you can see as discussed earlier we have references to the relevant packages here from within our gomod file and we have this gossum file which has also been created for us here since we installed the jinggonic web framework let's go back to the main.go file and let's write some basic code first of all you can see here that the FMT package reference here is underlined it's got a red squiggly line under it and that's just because we've imported it and we're not currently using it because we removed our FMT ln code so let's write the code for our Genonic functionality this is going to be very basic functionality that we create just to test that we've got the Genonic web framework now installed and is now ready to use so firstly let's just write some basic code using the Genonic functionality okay so let's firstly type routter like that and go rout equals so this operator means we are both establishing or declaring the type for this variable as well as assigning a value to it and we want to assign jin dot default to this variable and you can see here it's automatically imported gonic so it's automatically imported the jonic web framework package here and the the reason there's a red underline under router is because we we've declared and assigned a value to the routter variable but we haven't yet done anything with it okay so firstly we need to actually include open and close brackets here and now let's create an end point a very basic endpoint initially just for testing purposes so this is going to be a get a HTTP get endpoint so we go router.get open and close brackets and then let's create the root so it's going to be for slashhello like this and then let's create a function a function handler for our get end point so to do that we simply type funk like this open and close brackets and now we want to include a parameter within our function definition let's call this parameter C because it represents the context of our Jin Gonic web framework so this is the context of the incoming request coming from the client which allows us to call various functionality on this C object that is given to us through the function handler method automatically this is the function handler method for our for/hello endpoint great so we got C and then we include the type of C which is star jin dot context like that excellent and then let's implement the code for our function handler method so we also want to close the brackets here okay and all we want to do here is return C dot string return a string to the calling code so we're going to go C.string 200 which is a restful web API return value indicating that the execution of this function handler method has been successful and let's return a message hello magic stream movies like that okay and we have a little issue here and we don't need this bracket here that's the problem there and let's save that and now we need to run the router on a particular port so in in go we can do it like this we go if an assignment operator router.run and we're going to run our web framework on this port so we're listening on this port port 8080 here on the server and then let's make an assertion if error is not equal to null let's run this code which uses now I see that the FMT package has been removed we're going to need the FMT package here because we're going to write something to the screen so if error is not equal to null we want to write something to the screen and so it's going to be fmt dot print ln open and close brackets and a appropriate message failed to start server and then comma and the error like this and we need to include FMT here within the imports fmt excellent um and nil should just have one L here so let's save that and now our server our web API should be listening on port 8080 and we should be able to access through our browsers this/hello endpoint and it should return a string value of hello magic stream movies great so to run our code let's just save that make sure that that is saved first then type go run dot press the enter key great so it's telling us that it's listening on port 8080 so we want to go to localhost port 8080/hello and it should return hello magic stream movies to us through our get request and we've got our get request function handler method which is returning the relevant status the HTTP status of 200 as well as a message and that should be printed to our browsers so let's go into our browsers so I'm going to activate Chrome here so I've got Chrome running here and I'm going to type HTTP we're running on HTTP like this localhost and then we want port 8080 forward slash and we type the name of our route or the name of our endpoint hello like this and let's see if that works hello magic stream is printed to the browser as expected so our ginonic functionality in its most basic form is up and running for us to use so we've created a route and we've been able to access that route through our browsers and our function handler method handling the relevant functionality for our hello route has been executed and printed the appropriate text value to our browsers it's returned it to the client and printed the appropriate value to our browsers so great we're now ready to write more complex functionality and get into the substance now of the course excellent okay so now we are going to get serious about building our restful web API solution using Go and Jenonic so before we start writing our code let's firstly create a basic folder structure so that we can keep our code neat clean and dry of course dry stands for don't repeat yourself so we want to keep our code in discrete easy to find structures so that as our project becomes more complex we don't get lost when we need to for example implement a piece of reusable code right so firstly let's create a folder named so let's hit this icon here new folder and let's name this folder con controllers like that excellent this is where we'll store our controller code which will contain endpoint function handler code used for implementing logic for our HTTP endpoints if this isn't clear to you at the moment don't worry it will become clearer as we progress with implementing the functionality for our controllers we've actually already created a HTTP endpoint function handler that handled the functionality for the hello endpoint if you'll recall it was a very simple function that returned a message containing the text hello magic stream movies to the client and the next one we want to create is a folder called database so let's select that you can rightclick and go new folder and let's call this database like that we'll store code here that will leverage the MongoDB go driver to connect our web API component to our MongoDB database right so the next one is middleware whoops we don't want to create it there we want to create it here so let's select that folder right click go new folder and middleware like that press the enter key got our middleware folder we'll implement code within this middleware folder later when we implement the authentication and authorization functionality the middleware code is relevant because some of our endpoints will be protected meaning only logged on users will have access to these protected endpoints and conversely some of our endpoints will be unprotected meaning a user will not need to be authenticated before accessing the unprotected endpoints the meaning of the middleware code will become clearer as we progress with our course and we get into the authorization and authentication functionality later in the course so the next one we want to create is a models folder so let's select this folder here the root folder right click new new folder and let's call this models like that excellent this is where we'll define our models which are used to define the structure of the data that we'll be handling from within our web API solution the structure of the relevant models will be based on the relevant document structure seen in our MongoDB database for example we'll create a movie strct that will be based on the structure for a movie document stored within the movies collection in the MongoDB database so the movie strct will represent our movie model which is based on the movie document structure implemented within our movies collection within our MongoDB database great so the last one actually not the last one but the next one second to last one is called roots so let's create a roots folder within our root directory okay so here we go roots excellent uh here is where we'll write the code for defining both protected and unprotected routes essentially these roots define a path that client code for example JavaScript code running in the user's browser can use to access the endpoints defined in our serverside code our web API code written in Go as discussed the unprotected roots pertain to endpoints that clients can access without the need to be authenticated before accessing the relevant unprotected routes protected roots pertain to endpoints that a user can access only after being appropriately authenticated so the user will need to successfully log into the system before accessing protected endpoints we'll look at the authentication and authorization functionality in detail later in this course right so let's create the last folder and this is called the utils folder so let's create a folder called utils within the root directory utils so this is where utility functionality will be stored we'll store code here for reusable helper or utility functions that can be reused throughout our web API solution great so now that we've got our folder structure in place let's create our first file our first code file within one of our newly created folders and we're going to create the new file within our models folder so to do that let's create a file called movie model.go like this so we're going to create our movie strct within this file to access our MongoDB database we'll use a specific MongoDB go driver for this purpose the ID for our movies model will be of a data type defined within this driver so firstly we need to install the MongoDB go driver so let's do that okay so firstly we need to navigate into the same directory where the go.mod file is stored so we need to navigate to the server slash magic stream movies server magic stream movies server like that and now we just need we can install the MongoDB driver for go with this command so the goget command so it's goget then go do mongo db.org org slash mongo d-driver slashv2 slash mongo like that and let's press the enter key and that should install the relevant go driver the mongodb driver for go as always we need to be a little bit patient here while the installation process process takes place excellent so now if we look at the go.mod file we should be able to see the reference here to the new Go driver and where is it there it is so here's a reference to the MongoDB driver for Go package here excellent so we're ready to create our model now let's go to the movie model.go go file and firstly we want to define we want to declare the name for our package that these models will where these models will reside so we're just going to call this package model like this excellent so this means we'll be able to import the model package which contains which is going to contain the various strcts representing our data models we'll be able to import the functionality within this file easily within other Go files and you'll see how we do this in just a bit the package declaration at the top of a file defines the package name to which the file belongs a package in Go is a way to group related files and functions together files that belong to the same package can share functions types and variables note that only exported names starting with a capital letter are accessible from outside the package unexported names starting with a lowercase letter are private to the package so these packages can be imported by other Go programs or packages so the name of the package affects how other Go files or modules import and use its contents so this named package declaration helps us with the organization of our code so as our project grows in size and becomes more complex we are able to identify relevant code efficiently now let's write code to import the relevant data type information from our MongoDB driver for Go package you'll soon see that we need the BSON.object object ID data type to define our ID field within the movie model which is what we're about to create as a strct this bson object ID data type resides within the MongoDB forgo driver so let's create an import block like this so import like that then within round brackets we can import the relevant package so all we do is we type go.mongo db.org slashgo.mongodb.org slmongod-driver/v2 slashbson like that and the only reason it's got a red squiggly line under it is because we're not using the the driver functionality anywhere yet but we're about to do that so anyway let's press the enter key and let's create the movie strct so let's uh declare our strruct and we can do that with this code here type movie strct and that's how we define a strct and go open and close curly braces and we want the first field we want to include is to store a unique identifier for a movie so let's call this ID and then its type as discussed is BSON.object ID so the first field is called ID and it's of type BSON.object ID and this field is for the purpose of uniquely identifying a movie document or or movie data for a particular movie so we've got ID there and then the next field is IMDb ID like this and it's we're going to define this as a string like that okay just a quick word about the IMDb field an IMDb number often called an IMDb ID is a unique identifier assigned by the Internet Movie Database IMDb to a specific title like a movie TV show or episode or a person like an actor director etc so for example the IMDb ID for the Clint Eastwood classic western Unforgiven has an IMDb ID of TT 0105695 great so let's create the next field which is named title like this and this will of course store the title of the movie for example unforgiven the shaw shank redemption once upon a time in Hollywood etc and the next field is called poster path like this and it's also defined as a string this field is used for storing a publicly available URL that points to an appropriate poster image for each of our movie documents shout out to the movieb.org website we are using the poster images available on their website in our application for this purpose and the next field is the YouTube ID field so let's create a field called YouTube ID like this and it's also of the string data type this field stores a YouTube video ID that points to a trailer for each movie document stored within our MongoDB movies collection we are using the trailers on YouTube as a substitute for streaming the actual movies you'll see a bit later how we can use this YouTube video ID value to run an appropriate trailer for each of our movies from within our web application so let's include a field named genre like that and this field is going to be defined as a genre strct which we haven't yet created and we want an array of genres stored within the movie strct great so in this field we are going to store an array of genres each movie can be associated with one or more genres for example airplane would just have one associated genre in the array which would be the comedy genre unforgiven could be associated with more than one genre for example western and drama you'll see a bit later in the course how we are going to use this genre array to help recommend movies to the user so we need to create our own strruct definition for the genre field which will store an array of genre strcts so let's define the genre strruct like this type genre strruct and let's open and close the brackets like this and then let's create a field called genre ID and this will be of the int data type so the genre ID field will uniquely identify each genre within our database and then the next field is genre name and this is defined as string this field will store the actual name of the genre as a string for example comedy western drama thriller etc and then next we have the admin review field which we'll define as string this field stores the movie review associated with each movie document so an administrator which is just a user who is part of the admin role will create a review for each movie in natural language we'll use open AI through lang chain go we'll discuss lang chain go at the appropriate time and we'll prompt an LLM through the use of lang chain go to extract one word to define the sentiment behind each of the administrators movie reviews the sentiment can either be excellent good okay bad or terrible these options are specified in the relevant prompt along with the relevant movie review the sentiment can then be stored in a field which we'll name ranking so let's create the ranking field so the ranking field here is of type ranking and we haven't yet created the strct that represents the ranking type and we'll do that now so let's create the strct for the ranking data type so we do that by typing type and then ranking strruct open and close curly brackets like this and this has two fields one named ranking value like this and it is of type int so as discussed each sentiment excellent good okay bad or terrible has an associated value for example excellent has the associated value of one good has the associated value of two okay has the the associated value of three bad the associated value of four and terrible has a ranking value of five so the next one we want to include is ranking name and this is defined as a string data type so an integer value is stored in this ranking value field which is associated with the relevant ranking name field which as discussed can be one of the following excellent good okay bad or terrible so these fields are part of functionality that turns a movie review written in natural language into a quantifiable value a ranking if you like you'll see later how we can use these fields to help recommend movies to users so the next thing we want to do is define how we want our fields to look in BSON and JSON format so the BSON format refers to how the field looks within the relevant document in the database and the JSON format is how the data will look how the relevant field will be named for example when the data is returned from our web API application to calling client code for example to JavaScript code for example running in the browser so to do that we open and close back to characters like this and then for the bon format we type b then within quotation marks we include how we want the format to look id like that excellent and then we can also do the same for the JSON format so it's got to remain within the back characters and we type JSON and in this particular case the ID is represented the same way as the BON as in the BON format so it's underscore lowercase id like that so we can define within our strcts how the fields map to our MongoDB database as well as to the JSON data that will be sent to calling client code so for example our ID field must conform to this BSON format because this is how the field is defined within our MongoDB database and we want our ID field to look the same within the JSON data sent to calling client code this may not always be the case sometimes we may wish a field to look different in JSON format to what it looks like within the database therefore the BSON definition here may be different to the JSON definition here in some cases so we include these JSON and BSON definitions within back characters like this and let's fill out the other fields so bon and we want in bon format we want our field represented as imdb ID and in fact we want the same within the JSON format so we just include the same format here like that and we just really go through here and do the same sort of thing for each field so the title field bson in quotations title will just be title like this and you can see that in all of the cases so far bon and Jason are exactly the same what have I done wrong there underline bon genre id okay so we can't leave gaps like that or else it complains okay so interesting anyway let's just remove that gap and that s and let's include the definitions here so the last thing we want to do in this section of the course is create declarative rules for each of our fields we can establish within our strct rules for each of our fields so that only valid data is stored within our strcts so when for example client data is passed into our serverside code at the point where our code maps for example our movie strct to specific movie data passed in from the client the movie data is automatically validated based on the rules that have been defined so establishing these types of rules can for example protect our database from being updated with undesirable data so let's look at the IMDb ID field so for this field we want to include validation whereby this field cannot be left empty so validate so if the client passes in data representing a movie this field is required so it cannot be empty so we include the required keyword here within quotation marks like that so this validate keyword here is associated with a comma delimited string of criteria validation criteria represented by keywords like for example the required keyword like that so we're only going to include the required validation criteria for the IMDb ID field here okay let's move on to the title field and for the title field we want it to be required so we include the same thing as we did in the other in the IMDb ID field so we include required like this but we also want the title field to contain more than two characters and we can enforce that rule by using the min equals 2 code here within quotations and we can also include a max so we don't want this field to contain more than 500 characters so we can include max equals to 500 to enforce that rule oh still got a brown squiggly line there can't believe it oh it's cuz that needs a space there okay perfect so that now is formatted correctly so the the yellow squiggly line went away no more warning okay okay let's move on to the poster path field and we want this to be a valid URL so let's include validate colon within quotation marks we first want to include required this has to be filled in this has this poster path field cannot be empty and then comma and then we can include the URL keyword which means it has to be a valid URL for this path here okay required URL and we have problems here all right and it's because we haven't closed off poster path here okay and now it likes it it's okay all is well okay so within here validate and we're just going to include a required validation criteria here okay brilliant genre so for genre we want it to be required but we also want the validation to dive into the nested array so let's first include required here let's first include required and then dive this keyword dive ensures that the nested structure here which is an array of genres will also be validated and we'll include the various validation criteria in just a bit for the genre strct okay and this one here the admin review will just be we'll just include the required keyword like that okay and we'll just include required validation for the ranking field okay required right so let's move on to the genre strct and we've got as you can see here we've got the dive keyword so this will also be validated so let's include the criteria and this must be required and that's all we need for the genre ID okay and then for the genre name we'll include we'll include the required criteria but also min and max criteria so min equals to two whoops max equals to 100 lastly we just need to include the validation criteria oh and we haven't included the BSON and JSON definitions here either so let's do that first so so validate let's just make this required like that there and then let's include the declarative code for the the appropriate declarative code for the ranking name field so this is just ranking name ranking name it's a required field and now you could establish a little bit more complex validation for this field because you might want to include validation where only one of the following values can be stored within this field so to do that you can go comma one of so perhaps you don't need the required because you're already telling it that one of these must be in the field so we go one of equals excellent so this is delimited by a space these values excellent good okay bad terrible but you may want this field ranking name to be extensible because the way I've designed this application is that you could potentially create a huge spectrum of sentiments not just excellent good okay bad and terrible so you must bear in mind that this functionality can be be extended in which case you you'll have to change that here so in fact I'm going to remove this because it's a bit too restrictive especially at this point in the development process so I'm just going to make this required brilliant and that's it so in this part of the course we are going to create an endpoint and an endpoint function handler named get movies within a controller that returns movie data to a calling client we'll write code to query our movie collection from our MongoDB database and return all the movie data to a calling client we'll use the Jin Gonic web framework to help with HTTP requests and HTTP responses and we'll use the MongoDB driver for Go to query our movies collection within our MongoDB database we'll then send a HTTP response to the calling client that will contain a list of movie data but firstly let's get set up with the basics for our get movies endpoint function handler and then set up the endpoint that will make it easy to invoke the functionality in our get movies method via HTTP okay so we're firstly going to create a file within the controllers folder so let's select the controllers folder here and let's click this icon here to create a new file and let's call this file movie_ontroller so we are using a common naming convention here which is snake case for the name of our movie controller file excellent um and because I'm using snake case here we should probably use snake case everywhere to keep things consistent so I'm just going to rename the movie model.go file to movie and then a lowercase M here for model great so that's now consistent we're using the same naming convention for these files so let's go back to the movie_controller.go file and let's name the package so we want our package to be named controllers yep like that so we've got our models our model package here yes okay so this package we should have named models so I'm going to do that now i'm going to name this package models so that we're being consistent with our naming convention so this is a package named models which contains movie model and this is the movie_controller file and it's part of the controllers package great let's create a block for our imports like that right out of the gates I'm going to include the gingonic package.com so this is the package for the gingonic web framework for slashjin like that let's save it excellent and in fact that should be import not imports okay okay and this red squiggly line is just because we're not currently implementing any code regarding the G jinggonic web framework so let's implement a function called get movies and ultimately this function will be used to return a collection of movie data queried from our movies collection that exists within our MongoDB database and return that to the calling client code so let's create the function we use the funk keyword and we want this function to be exportable so it's going to the first letter must be capitalized if the first letter was not capitalized it means the function is a private function but we want this to be a public exportable function so we're going to capitalize the first letter here get and let's move these like that and you can see it's in camel case and let's open and close brackets like that and now this function must return a jin dot handler funk type because we're now going to actually return a function from the get movies function and within the return function will contain our implemented logic so let's type return like this funk open and close brackets C and then star jin dot con context like that and then open and close curly brackets and we're going to then implement sorry there should be a curly bracket here open the first function and then a closing bracket here to close the nested or the returned function here and you can see now those red squiggly lines have actually disappeared here because we're now implementing a function handler that uses Genonic this is how we're hooking into the Genonic web framework so here we are able to interact with the Genonic web framework through a function that returns a function the interaction with Jenonic is facilitated by our endpoint handler function that returns a gingonic type jin handler funk so basically get movies returns a function of type jin dot handler funk through this context object denoted by the C parameter name passed to our HTTP endpoint handler function get movies we are able to read incoming HTTP requests as well as create HTTP responses so let's create the relevant response so we're just going to create a test response for now and then after this just so that we can test our root we're going to set up our root for our endpoint and map that route to the get movies function handler so firstly we're just going to create a basic JSON response here so we're keeping this very simple just so that we can set up so return 200 HTTP status okay and then we can use jin.h H to help us return a valid JSON message to the calling client so let's go message list of movies of course in the actual end code we're going to be we're going to actually return a list of movies a list of movie data but before we do that let's just set up our route correctly so that we can access the get movies functionality through an endpoint so to do that let's go to the main.go method here and let's create a new endpoint let's go to do that let's go rout.get we're using an HTTP get request to access this endpoint this endpoint is going to be for slashmov and then we're going to map that to to our method so firstly in order to do that we need to actually import our controllers package into the main package and to do that we simply type the following controller and then within quotations and I'll explain what the controller keyword here represents in just a bit but let's include the path to or the name of our package the full name of our package which includes github.com/gavanlon digital slashmagic streamovserver slashmic check stream movies server and then forward slash controllers so this might look a bit weird to you at the moment but let's go back to when we created the go.mod file if you look here we named this github.com gavanlon digital magic stream movies server magic stream movies server so we want all of our packages to stem from this root path basically and this just makes it more futurep proof our code and ultimately easier for another developer to import our packages into their application so we're going to keep this path as our root path and then all the other packages that we create will stem from this so for example if we wanted to import our models you would just include this root path forward/models and the same for controllers so let's go back to the movie_controller file here and you can see that this package is named controllers so if we go to the main.go file we have controllers appended to this root path if you like so github.gavlon digital magic stream movies server magic stream movies server controllers and you can make sure that that's correct by just copying and pasting that so if you copy that go back to the main method to the import section here and you can just paste that in there forward slash controllers like that and we're doing this so that we can access the get movies endpoint function handler that we just created in our controllers package as mentioned earlier this get movies function is exportable because the first letter of the function name is capitalized if get movies had a lowercase G it would be deemed as private and therefore not exportable so because it is exportable we can access it from within the main method once the controller package has been imported so here we are importing the controllers package like this now you can see here we are using this import alias and this is just for readability so instead of accessing the get movies method via controllers we are going to use this import alias controller so that's why there's a red squiggly line under it because we're currently not using it and we're about to use it here great so now let's access the get movies function from the controllers package and we because because we're using an alias controller we can type controller here dot and in intellisense we've got get movies and that's the one we want so we are mapping the for slash movies endpoint to controller.get movies here and if we just go back to get movies we can select that and go to go to reference sorry go to definition okay and you can see we are just sending back a JSON object to the client message list of movies so we should be able to run this go to our movies endpoint and receive this message assuming that our route has been set up as we wish it to be set up and we're declaring how we wish it to be set up with this line of code here so let's run it let's first go into the root directory cd server CD magic stream movies server and let's type go run dot okay and it's listening on port 8080 so if we go to port 8080 we should be able to we want local host but we don't want that port we want 8080 not hello this time we want movies that's our endpoint if we click the enter key if we hit the enter key rather oops 88 that's not what we want we want 8080 there and there we go message list of movies so this means that our end point has been successfully mapped to the get movies method which is what we want so now we are in a position to query our MongoDB database and return the actual movie data to the client so let's get on with that then so let's go back to our code here and cancel out of that clear the screen and now that our route is set up let's complete the logic for our get movies function handler method right so I'm going to delete that so now that we have our for/mov endpoint and we've mapped it to our get movies uh function handler let's create the code that queries our magic stream movies database and returns the document data within the movies collection to the calling client but before we do that we need to create reusable code that will connect our web API solution to the MongoDB database and that will be done through the use of the MongoDB for Go driver let's create a file within the database folder here new file and let's name it database connection.go like this press the enter key package database so the code within this file will belong to the database package let's include an import block like this and we want to import FMT the FMT package so that we can write certain values to the screen this is good for when we're during the testing phase of our development during the testing and development phase and we want to be able to log information so let's include the log package and we want to include the OS package and this package will be used for reading environment variables which we will configure in a bit within a file called env and now very importantly we need to import the MongoDB for Go driver packages so if we go to go.mod we can see we've installed the mongo driver here so we can actually just import the packages we actually went through the install process already oh yes and the reason we've already installed that is because we needed the bison.object ID data type which resides within this package here that's why we've already installed it okay so let's go back to the main.go file sorry we want to go back to the database connection.go file here okay and let's include the following imports so we want go mongodb.org or slash mongo dashd driver/v2 slash mongo like that so go.mongodb.org/mongod org/mongod driver/v2/mongo and then we want a similar import here but this time we include a forward slash options and we'll see the package functionality in action in the context of our database package in just a bit and of course they have these have all got red squiggly lines under them because we're not using any of the functionality of these packages yet so we're going to create a function called DB instance which will be used for connecting our Jin Gonic web API application to our Mongodb database through the use of the MongoDB forgo driver we have already imported these driver related packages for this purpose okay so let's create our DB instance method so to do that we type funk DB instance like this open and close brackets and we want this to return a value of type mongo client like that let's open and close curly brackets and the first thing we actually want to do is read an environment variable so to read an environment variable from av file we actually need a particular package and we haven't yet got that installed it's called joo/go.env so to install it we use the goget command so let's type goget and then it's github.com/ slashjo slashgo.ev so go get github.com/jo/go.env let's press the enter key and it's done it for us excellent so we can verify that if we look at our go.mod file we should be able to see a reference to it here it should have added it and there it is great okay so let's create the code for our DB instance method so as discussed we want to access av file but we don't yet have an env file to house our uh various environment settings so to do that let's rightclick magic stream movies server here let's go new file and then create a file called env so the first environment variable that I want to configure is called database name so let's configure our database name here and we've called our database magic stream movies we can verify that let's just save that and we can verify that by going to compass let's launch compass okay so magic-stream.mmov that's what we've called it okay that's what we've called our database so let's go back to the code and change this to magic dashream dash movies like that and then the other environment variable that we want to create is called MongoDB uri so this is a pointer to our MongoDB database this is a connection string to our MongoDB database and we can configure it here and this will make it easier for when we deploy our application to the cloud later on and we can just configure the new location of our database here the new connection string for our remote deployment deployed version of our MongoDB database we can just configure that connection string here so let's minimize that and go back to compass and we can actually get our connection string from here like this so we can just click on this and copy the connection string to our clipboard so let's head to this option here and then we can then we can just paste it in here excellent okay and let's go back to our database connection.go file okay so the first thing we want to do is load our uh file that contains our environment variables into memory and we can do that with this line of code env.load and then in quotations env like this and oops and if a error has occurred while trying to load this file it means it might be deployed to a remote server that doesn't contain this file so we're actually just using our environment variables locally at the moment and that's why we're using this particular package the environment variables will be configured differently um when we deploy it to the cloud but we need this line of code here so in fact this line of code will return an error in the cloud so we don't want it to be a fatal error we just want to check for the error and we can just print it to the screen just print a warning telling us that the env file does not exist and this is this probably is harmless because it means that it's deployed to the cloud and it's reading the environment variable a different way we'll cover that in more detail when we deploy the application to the cloud so if error does not equal to null this means an error has occurred which probably means that the env file does not exist which probably means that the version that's running is a deployed version and the environment variables are not configured within av file but we'll discuss this a bit later so I'm just going to print a warning message to the screen warning unable to find envile like that great so if you had for example gone log.fatal here the code would actually stop it would print the the error to the screen the error message to the screen and this function would and the execution of this function would discontinue at this point so we want to print it as a warning message so it's not a fatal error great okay and then let's read the environment variable so let's include this line of code that assigns that assigns the relevant environment variable we're using the OS package here get env method and we want to read a particular environment variable we want to read the Mongodb URI environment variable so let's go back to that and paste it here and then we can check so if mongo db is equal to an empty string then some sort of issue has occurred and our application cannot really continue without this information so we're going to log a fatal error this time dot fatal we don't want the code to continue after this point so we go log.fatal fatal and we can output a message saying MongoDB URI not set not set that should do okay can't read the MongoDB URI which contains the connection string and we can just make we can just print that using FMT to the screen we can actually print the the relevant environment variable to the screen using the FMT package like this comma mongo db like that excellent so now that we've got our URI we should be able to connect our code here our client code in this case even though it's serverside code it's in the context of connecting to the MongoDB database it's client code so we can go client options colon equal options dot client open and close brackets dot apply uri and then let's include the readin environment variable which is mongod db excellent and then now that we've read in the actual connection string we want to actually connect to our Mongo database our MongoDB database so let's do that with this line of code client error you see this function that we're about to to call returns two values one which is a Clyde object and the other which potentially contains an error if the code hasn't been executed for some reason if the code could not be executed for some reason due to an error this object will contain a value great so now we call mongo.connect connect and we pass in the client options to connect to our MongoDB database so if error is not equal to null we'll cover error handling in just a bit we'll cover the details of error handling in just a bit let's first write our DB instance method let's just get the fundamental code in place first and we'll look at some of the details in just a bit so if it's not equal to null I'm actually just going to return null to the calling code in fact let's return the in fact so if is not equal to null let's just return error to the calling code this means the code will cease to execute after this and it will return whatever error occurred at this point in the code and if no error has occurred let's return the client object for now let's just return null if the error occurs at this point we'll just return null just to get things up and running we can handle the details of that exception a bit later so now we want to create a method which opens the actual connection to our database so let's create a function called open collection like this and this function contains a collection name parameter as a string so for example if we are querying the movies collection movies will be passed in here as a string okay and we want this to return a collection so mongo collection like that open and close curly brackets and we can implement the logic for this method now okay so we need to firstly load the env file to see if it exists and it will exist if the code is running locally but perhaps not if but won't exist if we've deployed it to the cloud so let's do a check for that let's go gov.load and let's load the env file here like that so if uh is not equal to null let's do the same thing we're just going to print a warning okay like that the env file doesn't exist and then we want to read the relevance environment variable at this point so and assign it to a variable called database name and you can see that we're using this special operator here colon equals which means it's actually declaring the variable and its type it's inferring the type by the returned value here so get the get env method returns a string data type therefore database name because we're using this particular assignment operator it knows that database name go knows that database name is defined as a string at this point so we don't have to explicitly for example in a line up here go something like string database name because it already knows that this is a string based on what is returned by this method okay let's go data base name like this okay and we can go use fmt print ln just to for testing purposes to make sure that the correct database name is indeed being read from the environment variable and we can just output it here like this okay and that's just for testing purposes great now we want to actually load the relevant collection so to load the collection we go collection and once again we are inferring its type by the returned value from a method which we'll write right now and we actually need the client object at this point so outside of all of our methods we want to create a client object based on the DB instance so the client returned from here will be our client object so let's call this client and it's of type mongo.client equals to the return value from our DB instance method here great so we can now use this client object to open a collection so let's do that let's go client database and we've read in our database name so we can use this variable which should contain our database name and pass it in to the brackets here and then dot collection and then we want to pass in whatever collection name has been passed into the open collection method so we can copy that and paste it in there like that if collection equals to null obviously something has gone wrong so we'll return null to the calling client code but if it gets to this point we'll return the actual collection this has only one L i keep giving null two L's sorry about that so that's now corrected so now we have our um open collection code written in this method here we are able to connect our client which was set up through our DB instance method here to the MongoDB database from within our web API component excellent so we're in good shape here and we're ready to write the code for the get movies function that will connect to MongoDB query the movies collection and return that data to a calling client via HTTP and we're going to do that with the assistance of the Jingonic framework so this is the context variable here and we're going to use that to help us uh return a response to the client okay so firstly we want to include this line of code here cancel and this is done so that no matter what happens in this method that our query that we're about to write will time out and clear up any resources that are hanging about so this is really just to ensure that housekeeping is automatically conducted no matter what happens in this method so context dobackground and we can set the time out here to 100 seconds second like that okay and you can see that because I'm referenced the time object here it included the relevant package automatically within the imports block here within the import block here okay and these are underlined because we're not yet using them these variables here returned from the with timeout method and we also need to include this line of code defer cancel like that and then let's define a variable called movies which will store an array of movie models a movie strcts so firstly we actually need to import our models package from here okay okay and to do that we can like we did here where we're importing the controller controllers package which is one of our packages we can do a similar thing here i'm just going to copy that and change the relevant bits so let's go to movie.controller movie_controller.go and include the local import here and instead of controllers this will be models like that so we're now importing our models which will contain a reference to our movie strct so we can go models dot movie like that and define our movie movies array as models.mov so this array can store a number of movie struts within it okay and then let's create a cursor for the purpose of querying and returning data from our MongoDB database so firstly we actually need to create we need to connect to our movies collection and because we've written all this reusable code here we can use this open collection method here to connect to our MongoDB database so we also need to import the database package into our controller file here movie controller file here so let's make a copy of that copy that paste it here and replace models with database like that there and then we can use the exportable methods within the database package within our get movies function handler method here so let's create outside of any method here let's create a connection to our movies collection and we can do that with this code var movie collection and let's And our movie collection is of type mongo.olction oops like that equals to and then we can refer to database because we're now importing our database package dot open collection which is our method that we created within our database package and we can pass in the movies collection here as a string so we pass in the movies name which is the name of our of the collection we want to connect to here excellent so it's running our code here to connect to the movies collection so let's go back to movie controller there and let's use our movie collection variable here which is been deliberately coded outside of the method so we can access so we can access it with any method within this file so let's paste movie collection there dot and let's call the find method and we're actually calling the find method on a collection object a MongoDB collection object so we're using the MongoDB forgo driver here and let's pass in the context ctx and then bson m like this and we need to create a reference to bson here in order for that to work so we've actually already done that in the movie model file so we do the same here we import the BSON package here okay and that's now working and of course we are not using any of these variables that's why we have these three problems here and these are underlined okay so let's move on to that the next code let's check the error object so if it's not equal to null and I will get it right this time null 1L okay we can use the context and write JSON to the response like this and we want to include HTTP 500 status error and we can use this value here for our internal server error so we're saying an internal error an error has occurred if the error is not null here an internal error has occurred within our web API component we're sending that to the client and we can use jin.h here to send back a JSON response error failed to fetch movies like that okay excellent now if it gets to this point so no errors have occurred there we'll use the defer keyword and I'll explain what this all means a little bit later and then we close our cursor so no matter what happens within this method the cursor gets closed and any of the resources that the cursor uses will be cleared up so this is really for memory management that we're using this line of code here and this line of code here of course okay so let's now create an if statement if equals cursor let's use our cursor here do all method called on our cursor let's pass in the context this is for resource management this context up here resource management purposes and then we want to pass the cursor into our movies array like that so we want to pass it in at this memory address so this amperand means that we're passing it directly into a particular memory address in memory into this movies array here so now let's put a semicolon there and if error is not equal to null we'll explain what the semicolon is in just a bit but let's just first write out our code okay so if that equals to null let's also pass back a status of internal server error okay uh and the message here should be failed to decode movies so it failed to insert the movies cursor oops we have a problem there failed to insert the movie's cursor into this movies array variable okay if an error is returned at this point here sorry no if an error occurs at this point here where we're placing the cursor into the movies array here great now if it reaches this great and now if it reaches this point here we can return using our context context our C variable which is the context object from Gonic passed into this method from the gingonic web framework we can use this C method the C variable the C parameter or argument to call the JSON method and pass back a status of okay so that' be 200 okay sorry status okay this must be capitalized and then we can pass the movies array and it will be passed down to the client in JSON format excellent no problems that's what we like to see and we can actually launch this and test this all so let's try that let's go to the terminal here and run our code go run dot and it's now listening on port 8080 so let's see what happens when we navigate to the movies endpoint let's see if we get any results okay let's fire up Chrome http localhost port 8080 slash movies look at that all our data has been returned and of course it looks rather ugly at the moment but when we create the React part of our code using Bootstrap 5 React Bootstrap we can make it look something like this i'll show you right now what we'll make it look like and there we go so this is what it's going to look like the end result will look like so it looks pretty cool our movies returned to the browser will look like this okay so let's write the code for another endpoint handler function that retrieves and returns data for just one movie this time from the movies collection in our magic stream movies database okay you can see here we've got the get movies handler function and now we're going to create a function that is for the purpose of retrieving just one movie rather than returning all the movies from the movies collection we're going to find one specific movie within the movies collection and that query will be based on the IMDb ID for the relevant movie okay so let's write the code so it's funk get movie like that open and close brackets and then we want to return a function just as we did in the get movies handler function we want to return a function of type handler funk like this and this is a way of hooking into the gingonic web framework okay uh so to hook into the gingonic framework we are going to return a function of type jin.andlerf funk so let's employ the return keyword here and let's write an anonymous function here so when I say anonymous function of course I mean it doesn't have a name so to do that we type funk open and close brackets like that and then we can pass in the context of the jinggonic web framework um through this parameter the C parameter here which is of type star jin.ext like that so this will give us context essentially from the web framework and will make it easy for us to well for example return a response to the calling client code or reading a parameter you'll see we are able to read a parameter from the URL which is something we're going to do in this particular function and handle requests handle request data and that sort of thing so the Gengoni web framework is making that a lot simpler for us okay perfect so let's write the code the first thing we want to do is create the context for our query to the MongoDB database so to do that we type ctx and it's returning two values so we also want cancel here and then let's use our assignment operator context dot and then of course the method with cancel and we want with cancel time out here and this is for the purpose of cleaning up resources okay within memory dot background and then comma and we wanted to time out in 100 seconds 100 times time dot second and we have a problem here uh it's not with cancel timeout it's with timeout that's the method we want to call here and then we need to include this line of code defer cancel like that as you saw earlier when we created the get movies endpoint function handler we are returning a function of type jin.andlerf funk which provides us a way to hook if you like into the gingonic web framework so this for example makes it easy for us to map a HTTP endpoint route to the relevant HTTP endpoint function handler this also makes it easy for us to handle HTTP requests and create HTTP responses from within the relevant endpoint function handler so you can see here we are calling the contextwith timeout method like this and this defer cancel code like this this code creates a new context ctx that automatically cancels after a specified duration so here the timeout will occur after 100 seconds the function returns ctx ct ctx is the new context that carries the timeout cancel is a function to manually cancel the context before the timeout if necessary defer delays the execution of cancel until the surrounding function returns this ensures that the context is properly cleaned up and resources are released even if the function exits early due to an error okay so now we're going to use the C argument here the jin frame the sorry the jin context or jin gonic context here to read in a parameter from the URL and we'll see the details of how this parameter value is passed in through the URL when we map this handler function to a particular endpoint and we're going to call the endpoint for/mov instead of for/movies which is mapped to the get movies function there but we'll do that in just a bit for now we just want to read in the movie ID which is going to be the IMDb ID for a particular movie stored within our database and then we can use the C object here param method like this and then pass in the name of the parameter we want to read from the URL from the end point path if you like imb ID excellent so that should populate our movie ID variable so the next thing we want to do is just check that we have in fact retrieved a value from the URL so let's create an if statement to do that movie id equals so if movie ID equals to an empty string let's pass back to the client using the C gen context object and the JSON method let's pass back to the client a HTTP status of bad request status bad request and then let's include a message jin.h and we can pass back valid JSON to the client like this so error colon and then an appropriate error message let's just say movie ID is required so no movie ID was passed in with the endpoint path so we need to fail this function call here and let's include the return keyword so that the execution of code discontinues at this point okay and let's now create a variable named movie var movie so this is a local variable of type mo models do movie and this is a userdefined type that we created a strct that we created within our models within our movie model.go go file here that we can use to store movie data great so let's go back to our movie controller and the next thing we want to do is check if an error has occurred while we attempt to query our movie movies collection with this with this code and we're using the find one method so we are employing the functionality within the mongo driver for go package as you can see here if we go to movie collection which this is what we this is the reasonable connection functionality we created in the previous part of this course here we are returning a mongo.colction collection and we're connecting to a particular collection through this reusable code here so if we go back to movie_controller we can see here that we are connecting to that particular collection the movies collection here so now with this code using find one we want to query the our database the movie collection specifically for a particular movie and the query will be based on a unique identifier which will be the IMDb ID retrieved from the IMDb ID parameter okay so find one and then let's pass in the context which is for the purpose of ensuring that resources are cleaned up so even if an error occurs within this function the relevant resources associated with the query will be automatically cleared up so this is a housekeeping mechanism that we need in place so that uh to prevent things like memory leaks for example okay so find one ctx bson m and now we're creating a filter here on the IMDb ID parameter value like this colon movie ID of course you can see here we've set we're setting and that shouldn't be capitalized sorry you can see here we're setting this local variable we're declaring it and setting this local variable to the parameter value passed in through the URL to the our handler function get movie yeah our handler function called get movie so movie ID equals to the IMDb ID parameter and we're using that here to query the movie collection for a particular movie okay okay and then we can then the next thing we must do is pop the ref pop's probably the wrong word um to put the relevant movie data if it's found by the find one method into our movie variable here so let's do that movie and we want to place it at a particular memory address of our movie variable so let's include an amperand preceding the movie variable name here defined as our movie strct so we want it to be placed inside a particular memory location here okay great and let's check if that error is null so if it's not null an error has occurred and we need to handle the error and all we want to do is pass back an appropriate message to the client and the relevant HTTP status and we can do that through the C.JSON method like we've done before so let's pass back the status http status not found so this means that the resource that we're looking for within our database was not found so we need to pass that back to the client with an appropriate JSON message and we can implement the relevant JSON message using jin h like this so we're employing the ginonic framework for this purpose here and so error Movie not found okay error sorry it shouldn't be a comma that should be a colon like that because it's a JSON type format here okay so we also want to include the return keyword because we want the execution of code to stop at this point then if no error has occurred with this through this line of code where we are finding the movie document based on the IMDb ID value then we can return that movie data through this line of code to the calling client code so we want to pass back a HTTP status of okay like this and then we can just pass the movie data back like that through this second argument here using the JSON method called on the C context object passed into our handler function our endpoint handler function through the Jinggonic web framework here brilliant you may have noticed the colon equals operator and you might have correctly inferred that this is an assignment operator where the relevant parameter value named IMDb ID is assigned to a variable named movie ID the colon simply means the type of the variable is inferred based on the value being assigned to the variable so colon equals is a concise way to declare and initialize variables in go it infers the type automatically it's only valid inside functions it's a Go idiom used often and preferred by local variable declarations right so we're now ready to map our handler function here get movie to an appropriate endpoint so to do that let's go to main.go let's just duplicate this line of code here firstly and change the appropriate parts so here we want to change this to movie because we just want to return a singular movie to the client code and we also need to change get movies to get movie here but the real difference between this routter.get get method and this one here is that we need to pass in a parameter value through the URL when this particular endpoint is called because the get movie functionality depends on the relevant IMDb ID value so to do that it's actually very simple we include forward slash colon and then the name of the parameter which is im db id like that so if we go back to our get movie method go to definition you can see here we are able to retrieve using this context object passed in by the web framework we are able to retrieve the IMDb_ID parameter value here and then we can proceed with our query and the functionality of this get movie method and ultimately return the movie data to the calling client so let's go back to the main.go go file and within the main method you can see that we have now created the relevant route for our get movie handler function so the only thing left to do is really is just test it so let's firstly go into the appropriate directory so that we can run our code so cd server slashmagic stream movies server like that and then we just type go run dot okay and and it's listening on port 8080 so we can navigate to our new endpoint through our browser so let's go to let's use Google Chrome let's use Chrome to navigate to the appropriate endpoint so http localhost colon at80 and this is the sort of thing we want we want to go to movie and then we need to include a particular IMDb ID and let's find a particular IMDb ID through compass so I think we should use this one let's uh query for the Empire Strikes Back star Wars: The Empire Strikes Back an absolute classic of course and so let's go here and include TT000080684 which is the IMDb ID for Star Wars: The Empire Strikes Back let's press the enter key and it returns the appropriate data we can actually verify this visually by copying the poster path here within our JSON data to another tab here and let's see if it returns and there it is look at that episode five the Empire Strikes Back so that is working brilliant okay let's find another one let's go to Compass and let's do Star Trek the Undiscovered Country so let's copy that ID to our clipboards and paste it in here so you can see how the IMDb ID is passed in through the relevant endpoint and there it is star Trek the undiscovered country let's copy the poster image here the path to the poster image which is a publicly available image address and let's paste that here and press the enter key and there we go star Trek: The Undiscovered Country and for good measure why not let's do another one and I love the movie Unforgiven so I'm going to copy this IMDb ID and paste it here let's see if it finds Unforgiven and there it is and let's confirm that by copying the poster image into the oops into the next t tab like this and what a movie this was rest in peace Gene Hackman but this was a fantastic Clint Eastwood classic and of course Gene Hackman was amazing in it as always morgan Freeman was fantastic and Richard Harris so what a cast excellent movie great so anyway we are absolutely getting there now i'm very happy with what we've done so far i hope you are too great so we've created two endpoint handler functions for two HTTP get requests one that retrieves data for the entire movies collection and one that returns data for a specific movie so we have covered creating genonic endpoint handler functions for HTTP get requests so let's create a handler function and let's name this function add movie like this and returns a type of jin handler funk because it's a handler function that hooks into the gingonic framework let's open our curly braces like that and let's include the return keyword here and then an anonymous function that will contain our actual implemented logic and we want to pass in a parameter named C of type star jin context like this and this is all for the purpose of hooking into the Jinggonic framework which makes it easier for handling requests and creating responses so here we we are declaring a function named add movie which will handle the functionality for an endpoint that we'll map to this function once we have written the logic for the function it returns a jin funk type which is just a type alias for type handler funk funk and in parentheses star context so this is the actual handler function that jyn or jin gone will call when a request hits the root it's attached to c star jin.context gives you access to the request response path forward/query params form data etc so our function is integrated with the Jinggonic web framework which makes it easy to access the request response path for/query params form data etc this function is going to handle a post request because our function will add movie data passed in from the client to the movies collection within our MongoDB database in our web API solution we are appropriately using HTTP methods like get and post we have so far implemented codes to handle get requests to retrieve resources from our MongoDB database and send them back to the client in an appropriate HTTP response using the restful architecture when adding a resource or resources to our database it is appropriate to use a HTTP post request so we are using the restful architecture in our Genonic web API solution rest or representational state transfer is a software architectural style used for designing web services and APIs it defines a set of rules and conventions for how clients like web apps or mobile apps and servers should communicate over HTTP a restful API uses standard HTTP methods get post put delete etc to perform operations on resources which are typically represented by URLs so our get movies and get movie endpoint handler functions hook into the gingonic web framework to handle HTTP get requests with our add movie endpoint handler function we are going to add a resource to the movies collection within our MongoDB database so we are going to use a post request we are going to map the add movie endpoint handler function to an endpoint that handles a post request so let's add the code to handle the context of our interaction with the MongoDB database so it returns two values from this particular method one which is the context and the other one which allows us to cancel the action that the context will be mapped to so we need to call the context dot with timeout method like this pass in context dot background like this and the amount of time we want to lapse before the action that this context is mapped to is forced to stop so we want that time lapse to be 100 seconds and we can implement that code like this okay great okay that looks better contextwidth timeout um okay excellent so the red squiggly lines under these two variables is because they are currently not used so the next line of code we want to include is defer cancel like that excellent okay so that red squiggly line has gone away and then we'll use the ctx variable a little bit later so as we did in our last two endpoint handler functions we implemented code using go's context package to create a timeout context and it's typically used in Jin and other Go web frameworks to control how long an operation is allowed to run before being forcefully cancelled this is useful for setting a deadline on longunning operations like database queries HTTP requests or any blocking operations when the timeout expires the CTX or context is cancelled and any operations listening on that context will stop context.background is the base context often used at the top level of an application or request it's not cancelellable by itself so we wrap it with a timeout context cancel is a function that manually cancels the context before the timeout defer cancel ensures that when your function exits it frees up resources associated with the context it's important to call cancel to prevent context leaks for example go routines waiting on a context that never gets cancelled note that a Go routine is a lightweight thread of execution in the Go programming language it's one of the key features that makes Go well suited for concurrent programming go routines allow you to run functions concurrently they are much more lightweight than traditional threads creating thousands of Go routines is feasible and common in Go unlike threads managed by the OS Go routines are managed by the Go runtimeuler which multipplexes thousands of Go routines onto a small number of OS threads you start a Go routine by using the go keyword before a function call so note that even though you haven't seen me use the go keyword explicitly in this web API project the R.Run yeah we've got rout.run the r.run method call starts a HTTP server using go standard net http package the net HTTP package itself spawns a new go routine for each incoming HTTP request so every time a client makes a request it is handled concurrently in its own go routine so let's look what happens under the hood jin is built on top of Go's net HTTP functionality the HTTP listen and serve function used internally by R.Run run listens for connections and calls a handler in a new go routine for each request this allows multiple requests to be processed concurrently and efficiently okay let's go back to our add movie function so let's create a variable for the purpose of storing the movie data passed in from the client in memory on the server okay var movie models do movie like this so this is the userdefined strct that we created earlier on within the movie_model.go file here so this is the type that we are going to use or the strct we are going to use to store the incoming client data in memory on the server excellent and it's just got a red squiggly there because declared but not used same with the ctx at the moment okay let's press the enter key so here we have created a variable of type models.ov and this variable will store the relevant movie data passed in from the client the models domov type is of course our userdefined type that we created earlier let's write code to attempt to save the movie data passed in from the client to our add movie endpoint and we want to save this data within our movie strct so let's write the code for this so then our assignment operator here we're just checking for an error returned from C.ind we want JSON so it's C shouldbind JSON like this and then the amperand character because we want to locate a specific part of memory so we use the amperand for this so it's a pointer to a memory location when we put an amperand it's a reference to a memory location rather than the data itself it points to the data amperand movie like this and then we're going to check the error by adding a colon there and going not equal to null so if the error is not equal to null we need to handle an error because an error has occurred and so let's type c the context the jonic context JSON method like this and we want to pass back a status of bad requests so status bad request like this which I believe is represented by the number 400 or 403 no it's 400 I think http status bad request jin and this is just a shortcut for creating a JSON object so jin.h and we're using the gingonic framework for that purpose so error and then let's map the key value pair the key error to a value of invalid input whoops input like this okay and then we want to include the return keyword so it will stop execution at this point because an error has occurred and we want to just handle that error send back an error message to the client and then terminate the execution of the function the functions basically failed at this point so let's just go over this code again quickly so C is of course the jinggonic context object which represents the context of the current HTTP request should bind JSON and then in brackets amperand movie attempts to pass the incoming JSON payload from the request body into the movie strct if there's an error during binding for example the JSON is malformed or doesn't match the expected strruct fields the error variable or the error variable will not be null so the code inside the if block will execute if there was an error this line returns a 400 bad request HTTP response jin.h error invalid input is a shortcut for creating a JSON object with a key error and a value invalid input the return keyword ensures the function exits immediately after returning the error response no further processing occurs this code safely handles JSON input by validating and binding it to a go strruct movie if the input is invalid it sends a clear error response and holds further execution the amperand character in amperand movie is the address of operator and go this gets the memory address of the movie variable i.e it passes a pointer to movie not the actual value so now remember we included the declarative tags in our movie model here for validation purposes like validate required so to validate the movie data now stored within our movie variable we need to firstly install a go package named go playground so to do that make sure that you're in the current directory uh rule of thumb is make sure you're just in the same directory as where the go.mod file has been stored and then we type this command to install go playground so it's just go get remember we're using go playground to validate our movie variable so whatever the client has passed into the add movie method we're validating based on this criteria here that we included within back to characters here and we need this particular package to perform the validation so we're installing a validation package commonly used in Gengonic applications so we type this command to do that go get and then it's github.com slash goy-N playground hyphen valid day tour slashv10 like this and let's press the enter key and something is happening Brilliant that's a good sign so we've installed our Go Playground package and we can verify that by going to go.mod here we should be able to see it somewhere here there it is validator our playground packages the references to the playground package are here brilliant so let's go back to our code and let's include the relevant validation code so to do that let's type if assignment operator validate but you see we haven't got this validate object set up yet so we need to do two things we need to import the go playground package firstly so we can do that here like this within quotations okay so let's type github.com slash Whoops github slashgoy-enplay whoops gosh go hyphen playground slash validator slashv10 and we're importing our go playground validator package like that and then the next thing we want to do is create the validator object and we can do that with this simple line of code we type var validate attor equals to validator dot new like that great and now we can use our validator which is which is provided to us by the go playground package here and we can use it now to validate our data great so validate we've called We've called the object validate and then dot hang on oh we've called it validator so let's call this validate okay see if we can get some intellisense now we've got the name correct validate dot strct then we pass in movie like this and then we can include a semicolon and check the error so if the error is not null we know an error has occurred and then we can handle the error appropriately and we want to send back an error to the client with a HTTP status of internal actually no we want it to be a bad request so it's a bad request 400 we're going to send back and then we'll use Jin Garnic or Jin to create a JSON response error response oops we type error colon and our error message could be the following validation failed oops failed comma and details comma dot error so the actual error message so hopefully this can be useful to our client code in the event that an error occurs at this point in the code okay great and then let's use the return keyword here to stop execution of the logic within the add movie function here okay brilliant so if we get to this point in code we want to then insert the actual data within the MongoDB database not sure why we're getting all this okay before new line missing comma before new line what oh I haven't included the if here sorry having a bad day the go playground validator is typically used in a genonic web application to validate a strct like movie after it's been bound from JSON it uses the Go playground validator v10 package a powerful and widely used validation library in Go validate is an instance of the validator validate.ruct movie checks the movie strct against any validation tags for example validate required on its field if any field fails the validation it returns an error not null and the if block runs this code ensures the movie strct is valid before proceeding the validation is based on our declarative validation instructions that we included between backtick characters for the movie structure for example if we go to our movie models file and we check some of these we've got validate required we've got min max we've got limitations on the minimum amount of characters we want stored in the title and the maximum amount of characters we want stored in the title field here we've got a URL validator here for the poster path so we can include declarative code here to create rules for our fields within the movie strct so our validation functionality here will validate the rules that we've established between backtick characters for each of the fields within our movie strct basically and then we are handling that error here okay excellent right so if any field fails validation based on its strruct tags the API responds with a 400 status and a helpful error message so let's write the code that actually inserts the data passed in from the client now that it's been validated we want to insert it into the database so let's write the code for that so we type result so two values can be returned the result and an error if an error occurs so let's assign that to the actual action and we want to use the movie collection and then we want to call the insert one method to insert a resource into our database a movie resource now we can map the context that we created up here we want to map that context to this action here context and then we want to insert movie into the database so we do this and then we let's check our error so if error is not equal to null oops so if error is not equal to null c.json and let's pass a relevant error back to the client so let's pass a status of internal 500 to the client internal survey error like this and let's use jin.h here to pass wellformed JSON back to the client let's include an error key and its value failed to add movie great that looks pretty good and then include the return keyword to stop execution because it's failed at this point so this go code snippet is inserting a document a movie into a MongoDB collection using the MongoDB go driver this is a MongoDB collection object connected to the collection where you want to store the movie data movies the insert one function inserts one document into the collection ctx is a context used for timeout cancellation or passing metadata movie is the go strruct or map that will be converted into a bon document and inserted into the MongoDB movies collection insert one returns a result which contains information about the insert operation such as the inserted document ID is an error object which is nil if the insertion was successful if insert one fails for example due to a connection issue bad data etc it responds with HTTP500 internal server error and a JSON object with a error message failed to add movie then returns to the client the return keyword halts further execution so this code tries to insert a movie into MongoDB if successful the code execution continues if it fails it sends a 500 response with an error message lastly if the code reaches this point we want to send a success message to the client a HTTP success message so we type C.json and then within parenthesis HTTP status created and then we want to send the result back to the client great so that is pretty much our logic finished for the add movie function so the next step is to map the add movie handler function to an appropriate endpoint an appropriate route so let's map an endpoint route to our new add movie endpoint handler function so to do that let's go to the main method and let's just make a copy of this here this code here but this time we want to use a post HTTP method and then let's map our handler function method to a end point and let's call this one forward slash add movie like that so that's the path to our endpoint that we'll add to the base address of our web application and then we go controller add movie like that and that's it we've mapped our add movie handler function to the add movie endpoint here so let's test our new add movie functionality but now it's not so easy to test this functionality as we did for the get movie and get movies functionality the get movie and get movies functionality are both mapped to HTTP get requests which makes it easy to just invoke the relevant endpoints through the browser the results are returned and we can see the displayed results within our browser in JSON format however we need to map our add movie handler function to a HTTP post request where we need to pass movie data in JSON format through the body of the relevant HTTP message so rather than for example create our own HTML form on the client that can run within our browser for testing the add movie functionality an easier option is to use a free tool called Postman for this purpose this is a great tool that we can use for testing our restful web API endpoint so let's navigate to www.postman.com right so let's navigate to www.postman.com so this is where you can download this free tool called Postman and we use this tool to test our web API endpoints then you want to click on this menu option here let's just accept all cookies and then we want to click on this header menu option here and then click download Postman so that we can install our free version of Postman and you can see here it's detected my platform Windows 64 and then please just go through the relevant instructions download the relevant install file double click on it once it's downloaded to your downloads folder and go through the install process and this is what Postman looks like okay so now we want to test our ad movie endpoint using Postman so I'm going to invoke Postman here brilliant okay and there we have it so this is what we want we want instead of get we want post to be selected here and then we've got the path to the ad movie endpoint here okay in order to test our result and then we want to go to body here and then raw like that okay and let's just minimize that let's go to let's go back to our browser here open up a new tab and I'm just going to go to GitHub here because you can download the relevant JSON data that we're going to use to test our post request here from this location we've got a file called add test movie doc we're going to add Highlander 2 one of the worst movies ever created we're going to add to our database that's just my opinion horrible movie i really loved the first one especially when I was a teenager but this movie diabolical anyway I'm going to copy the data for it and then we can just so you can copy it from this location from my GitHub repository Magic Stream GitHub repository copy it and then paste it within the body of our Postman request here so it's body raw and then copy it into this text area box here copy the data from GitHub here then of course we need to make sure that our serverside code is running our test serverside code is running so let's go back here and to run our code we just type go run dot okay and it should be listening on port 8080 excellent and now through Postman we can test whether we can add Highlander 2 which is an absolute abomination to our database okay so let's do that let's press the send button to send this data to our add movie endpoint let's see what happens 500 internal error oh dear what's happened there undefined validation function min okay let's go to our movie model here and let's check genre name min equals to two max equals to 100 okay so let's take out the gaps here i think it is a bit funny with formatting here it's a bit temperamental with formatting and let's save that let's try again go run dot enter okay it's listening on 8080 there let's send that through 400 bad request okay well we've gone a little bit further validation failed key movie admin field validation failed on the required tag okay okay because we got no right okay so so we've already said that the ranking is bad this movie was really really bad so our field validation is a is working that's what that proves so we go to 400 bad request which is correct because we hadn't included any field value for the admin review field here and now we have So this should work let's send that through 201 created so we've created Highlander 2 within our movies collection okay i'm not too happy with that oh right okay right all right so Highlander 2 has been created within our movies database let's double check that okay reload data and let's see if we can find Highlander 2 there it is but that doesn't look right there so I'll have to look at that but it has created the has created a resource but it hasn't created a unique identifier as I would have expected so I need to just have a look into that okay so the reason why that ID within compass was just a series of zeros and not what I would have expected to see for example a unique ID like this is because I hadn't included a particular keyword within the back to characters here so we need to include this keyword here omit empty there in the bon and I'm also going to include it in the JSON so if we include that in the bon the bon criteria here it will insert a unique identifier within the database if the ID field does not have an does not store a value when it is inserted when that data is inserted into the database so this will create a new unique identifier like this by default for us okay so let's take that there same code omit empty like that and place it here also like that there and now if we test our code again let's type go run dot to run our code okay great it's listening on port 8080 let's go back to Postman we've still got everything set up here and ready to go i actually deleted the previous record offscreen i deleted the Highlander that we had here so we're deleting a new So we're inserting a new record into the database and it should include a unique object ID for this resource here so let's test that let's press the send button okay we've got a 200 created status sent back to us which is great and inserted ID and that's what I would have expected to see so it's created a unique identifier for our resource within the MongoDB database so let's check that if we go to compass and let's refresh let's reload data and there we have Highlander the quickening and it's got a an appropriate unique identifier now included here for the ID field excellent so our code is now working i'm actually going to remove this particular validation from admin review because initially when we create the resource we actually don't need this to be to have a value so I'm going to take this validation away from this field here and you'll see why I'm doing that in just a bit when we create the the code for using the AI functionality to read the admin review and extract the ranking from that admin review using AI and we'll see how that works in just a bit excellent we're in good shape okay so when a user first accesses our Magic Stream website on the homepage they will see some of the movies that are available to be streamed however when they click one of the movie items on the homepage they will be prompted with a login screen for example we click on Shaw Shank Redemption here we are prompted to log in because we can't use their services we can't use the services provided by Magic Stream without authenticating first so we have to first log on so I've actually seeded the users collection with a user named Bob Jones so if I go bobjones@hotmail.com that's Bob's email address and I enter his password and I log in we can now we can now stream the Shaw Shank Redemption and this is just a trailer played through YouTube so it's not obviously the actual movie the gun and then stop to reload so So if we go back here so now we're we're able to actually stream the movies and of course I'm just playing the trailers through YouTube to that acts as a placeholder for the actual streaming service great so let's log out anyone can see the actual movies provided by Magic Stream but if you want to stream one of the movies you have to be authenticated the user must firstly sign up or register for a Magic Stream account the user must firstly be identified by the users's registered details the user's credentials before the user is able to make use of the services offered by Magic Stream so the user must log in firstly to the website and be authenticated based on the user's authentication details which will be the user's email address and a password that the user chooses when registering with Magic Stream so if we log in here we don't have an account we can click this link here to register and we have this registration form presented to us so we can also access the registration form through this button register and we can register our relevant user details and our credentials for example our unique email address and a password here so the first step in creating this authentication functionality is to create a model representing user registration details so let's go to Visual Studio Code and let's create within the models folder here let's create a new file and let's call this user_model.go so this is the file user_model.go that we're creating in order to include a user strct which will represent a users model a user model that will contain the users's details the user's authentication details and general details even though we are in a different file to the movie_model.go file where code within the movie_model.go file is part of the models package we can also make any code we write within the user_model.go Go file part of the models package simply by including this declarative code package models like that and you can see we've got the same declaration at the top of the movie_model file here package models so anything we write in here will also be part of the package part of the package named models which will make it easy for us to import whatever exportable code we have here into other packages and other applications okay great so all exportable code for example a model like the movie model where its name starts with a capital letter can be imported and used by another application or from within another package for example if we go to the movie controller.go file the movie_controller.go file we can see here we are importing the models package so we have the root path here which we established within the go.mod file by naming the module with this path on GitHub that where we will upload our code to eventually so in movie controller we have that as our root path the name of the actual module and models is the package name that we are the package that we are importing here so that we can use for example the movie model within our code so when we create our user strct we'll created it pretty much in the same way the same basic way that we've created the movie strct so firstly let's create an import block here and then we'll create our user strct okay and we need to import the go mongodb.org slashmongo-driver/v2 slashbson package like this and it's got a red squiggly line because we're not yet using the relevant functionality within this package okay so let's create our user strct and we can do that by typing type user strruct like this oops and then within the curly braces we're going to create our first field which is an ID field that uniquely identifies a particular user within the users collection and we'll type this as bon.object id excellent so the reason we need to import the go.mongodb.org/mongo-driver/v2/bent or/mongo-driver/v2/bon package is so that we can make the id property of the user strct of type bson.object ID when a new document is added to a collection in our magic stream movies database a unique identity is automatically generated for that document so this is what this ID field represents in our user model a field that uniquely identifies the document in the collection so this is a field that uniquely identifies a user resource in the users's collection let's also create a field that uniquely identifies the user so let's create a field named user ID for this purpose so user ID and this is of type string this field is of type string let's create a first name field like this of type string and a last name field like this also of type string let's create a field named email and this is also of type string let's create a password field also of type string note that we will include hashing functionality to obuscate or hash the password before it is entered into the database it is of course not good practice in terms of security to save the password as is to the database let's create a field named RO and this is also of type string so this is also of type string but must either contain a value of admin or a value of user we're only going to have two roles available so we could extend the role to enable multiple types of roles to be stored in the ROS field but for now we are going to restrict the values to either admin or user for this field of course only administrators will be added to the admin role which will mean they will have administrative privileges to for example in our application to add reviews to movies a user is only able to stream movies and in our application will not be able to review the movies i've deliberately made it like this for simplicity but you can of course extend the functionality so that the users can enter reviews for the movies so for auditing purposes let's create a field named created at like this and this field is of time dot time like that and you can see that we've got a red squiggly line here undefined because we need to import the relevant package and we can do that here like this and there the red squiggly line has disappeared then let's update our model with the update at field which is also for auditing purposes and this field is updated when a user's details are updated for example if the user's password changes etc so let's create that field so update at and this is also of type time dot time like that brilliant then let's create a field named token and this is of type string we must have it with a capital letter because this is exportable code so token string like this excellent we are going to use JWT or JSON web tokens for authentication and authorization purposes in this application so we are going to persist the values for the relevant tokens within the document that is represented in code by this user strct a JWT JSON web token is a compact URL safe token used to securely transmit information between parties as a JSON object it's commonly used for authentication and authorization in web applications a JWT has three parts separated by dots a header part which specifies the type of token and the signing algorithm for example HS 256 payload contains the claims user data or metadata like user ID roles or expiration time signature verifies the tokens authenticity using a secret key or public private key so let's look at how JWT is used for authentication a user logs in with their credentials the server verifies the credentials and generates a JWT containing user information the JWT is sent back to the client usually stored in local storage on the client or a cookie we're going to firstly look at storing the the JWT in local storage but we'll discuss why this is not the most secure solution and then we will eventually store it in what's known as a HTTP only cookie which means the cookie cannot be read on the client by JavaScript code making it more secure and less and less susceptible to XSS attacks we'll look at the details of this a bit later for future requests the client includes the JWT in the authorization header this is in the case where the token is stored on the client and then is added to the authorization header when for example a request is made to a web API endpoint but we are we we'll first initially look at this technique but as I said it's not the most secure technique we'll eventually we'll implement our code using HTTP only cookies which is a much better and more secure way of passing the tokens from client to server for example the server verifies the token signature and grants or denies access based on the embedded claims the benefits of using a JWT are as follows stateless no session storage needed on the server scalable easily used in distributed systems can be passed between systems and services securely then it is good practice to also create a refresh token so that the client is able to authenticate without the user needing to log in again once the token's expiry date is reached we'll discuss this further later in the course but let's for now just include a refresh token field like this and this is of type string just like the token field and the last field stores an array of our userdefined type genre which is based on our genre strct that we created within our movie_model.go file and you can see it here with all its validation that we also added in these tags here and we're going to use this within our user_model.go file and the reason we can just use it straight away is because this code here is part of the models package just like the code in the movie_model.go file so we got package models and we got this declaration at the top here so we can just use that strct automatically here because they're part of the same package so let's do that i'm going to call this field favorite genres like this and this is an array of genre strcts so we can just automatically include this type here because our code here our user strct is part of the same package even though they're in separate files even though the genre strct is in a separate file to the user strct we can access the genre type because they are part of the same models package so it stores an array of genres so when the user registers to use our magic stream movies application the user will be presented with a list of genres and the user must select one or more of these genres to provide our application with insights so that our application can recommend movies to the user that the user is most likely to want to stream we'll discuss our AI recommendation feature a bit later in the course so let's quickly look at the registration form for the prototype app that I created when I while preparing for this course so you can see here when the user registers with the system they can select their favorite genres and this will become part of the recommendation functionality that we'll add a little bit later where um this whatever they've chosen here will be instrumental in how we recommend movies to a particular user so I mean if we log on as Bob Jones for example joneshotmail.com like that there if we log on you can see that if we go to the recommend recommended tab here it's recommended five movies and in part this is due to the genres that Bob selected when he registered with the application so you can see for example he's more than likely uh when he registered he selected comedy sci-fi thriller for example or drama or fantasy and you can see it's ordering the recommended videos in terms of sentiments so excellent and you can see okay so the rest are okay so it's ordered by excellent down to the okays here but if for example if Harry Potter had a bad sentiment that would be displayed after airplane which has an okay sentiment so this is how we are going to recommend the movies to specific users but we'll look at the details of this in just a bit it's quite an exciting part of the course because we're going to be using AI to extract the sentiment from the movie reviews but we'll see the details of this a little bit later okay so let's minimize that let's go to let's go back to our code here and we want to include validation code for each of these fields so firstly let's open our back tick characters here oops okay and now for the JSON we want the field to look like this so it's just underscore ID so in the JSON that gets sent back to the client the field will be underscore ID okay and then we're going to include this omit empty keyword here and then we'll get back to what that means in just a bit so these need to be wrapped in quotations here and then let's address the BSON format so let's include BSON here and this is the way the actual field is represented within the database so it's underscore id the field will be underscore id like that and now if you'll recall we need to include this omit empty keyword so if for example just these fields are included in the client's payload which is what we're going to do so it's just these fields and no ID is specified mongodb based on this keyword here will automatically create a unique identifier for the relevant user within the database and that's what we want and that's the reason we're including omit empty here right so let's include the let's include the back characters next to user ID and we want so let's say the JSON we want the JSON for the user ID to be user ID like this okay and the BSON will be the same as the JSON in this particular case so we want within the JSON we want within the BSON we want the BSON field in the MongoDB database to also be user ID like that brilliant okay let's include the back tick characters next to the first name field so Jason we want the JSON to look like this first underscore name and the BSON is the same so in all of these fields the BSON and the JSON are going to look the same it's just the way we're setting it up but they could look differently if we chose to do so and then let's address the validation for the first name field so include validate keyword here colon and then within quotations let's include the required the required uh keyword like this comma min equals 2 so at least two characters must be stored within this field and max 100 so we don't want more than 100 characters stored in this field so that's our validation for the first name field and let's just make a copy of that because the last name field is very similar to the first name field and let's just paste that here for the last name field like this and let's just change the bits that we have to last name last name and the validation is exactly the same required min= 2 max= to 100 and then the email field let's do that so let's include our oops let's include our JSON criteria here and we want the JSON field to be email with a lower case like that and then the BSON we want also to just be email like this oops like this i mean okay and then um let's include the valid date keyword okay this is a required so there must a user must provide a valid email or must provide an email that's what the required keyword signifies but it must be a valid email and this is what email signifies so always remember that I think gaps may introduce an error so we must remove the gaps like this so these two must not have gaps just be comma delimited like that great okay and then for the password let's include the declarative rules for the password so JSON password will just be oops will just be password like this okay vson also just be password like that and then validate will be required and let's just keep this simple for now the password must have a minimum of six characters okay so very simple for the password and then the ro so this one will be a little bit more different to the other fields and you'll see why in just a bit so JSON and we want this to just be called roll like that and then bson going to be the same roll like that and then validate keyword now this is interesting so we want the validation to validate that this value is either admin or user so to do that we can use the one of keyword like that one of like that equals admin space so this is space delimited for the one of keyword or the one of criteria admin and then user like that so it can only be one of these two values included within the role field because we only have two roles currently in the system admin or user okay so the next one is created at and we'll just keep these ones quite simple so we just go JSON and then let's say created at like that and then the BSON is very similar to the JSON bon whoops bson created at and that's all the validation we want to include there let's make a copy of that so no validation but we want it we want the JSON to look like this and the BON to look like that and then for this one we obviously need to change the way the field will look in JSON so updated at all lowerase updated at all or lowercase for the BSON so the BSON and the JSON once again are exactly the same okay and then for the token I'm actually not going to include any validation for the token so we'll just include the way the token field should look in JSON and bon token and that's pretty much the same oops so this doesn't have back tick here we need to put a sorry back tick there and then of course we need to name this appropriately refresh token copy that refresh token for the bon and then here for the genre this also requires special sort of validation because we're actually going to be diving into the relevant genre strcts that will be stored in the favorite genres field here so the validation of importance here is the validation that we created on each of the fields within the genre strct so let's firstly include what the JSON will look like or the JSON sent back to the client so we want this to be called favorite genre all lower case favorite genres all lowerase and then the bison will be the same so this is how it will appear in the MongoDB database bson like that there so this field can't be empty so let's include required then comma now this is the important part dive so it dives into the value stored within the array and it val validates each genre which the validation criteria has been set up of course not there here that's the validation that will be used to validate the genre the various genres stored in the favorite genres array so just to go over this again the omit empty tag is used to indicate that a field should be omitted when serializing if it has the zero value for its type so also remember when we tried to add a movie to the movies collection in our MongoDB database the underscore ID field was set to a series of zeros because the relevant ID field did not include the omit empty tag this keyword omit empty will ensure that if when we add a user resource to the users collection that a unique ID is automatically provided for the newly added user resource to the ID field so when we add a client payload for a user's resource we're only actually going to be including these fields so we won't include a value for the ID field so we need this omit empty tag to be present so that MongoDB will automatically add a unique a unique identifier to the relevant user resource within the MongoDB users collection great and that's it we've created our user model excellent great so now we have created the user model let's create the code for inserting the users's details entered during the user registration process into our Magic Stream Movies MongoDB database once we have finished the code for inserting a user's registration details into our MongoDB database we'll test the functionality with Postman i feel that in the previous sections of the course we covered quite a lot of theory and have gone over a few detailed explanations regarding the code we are writing so this part of the course is going to be more about implementing code and we are going to move at a fairly fast pace here so that we can get the code written and tested for adding a new user to the users collection within our MongoDB database right let's get going right so let's add a file to our controllers folder and let's call this file user_ontroller.go at the top of the file let's declare the code that will be added to this file as part of the controllers package so let's type package controllers at the top of this file like this excellent so now the code within the movie_controller.go file is part of the controllers package 2 you can see we have declared that the code within this file the exportable code within this file is part of the controllers package and we're going to and we've done the same within the user controller file here so any exportable Kobe we write here will also be part of the controllers package let's include an import section like that let's import the following packages so let's import this package gethub.com/gavlon digital slashmagic stream movies slashserver slashmagic stream movies server for oops forward slashmodels like that okay and it's got a red squiggly line because we're not yet using the relevant model within our code which will be our user model okay great and we can just check that that path is correct by looking in the go.mod file and you can see the root of that needs to be the same as this here the root of that path needs to be the same as the name of this module so if we go back there we could actually just do this paste that in there and then for slash models to make sure that that's 100% accurate okay okay we are going to need a reference to the Genonic web framework package so let's include the gingonic web framework package within our import section like this github.com/jin dash Whoops gonic/jin like that excellent and we the red squiggly line is appropriate at this point because we're not using any of the functionality quite yet within the Genonic package we are going to want to validate the user registration data passed in from client to server so let's include the github.com go playground validator v10 package okay so github.com/go-playground okay sl validator slashv10 great excellent and it's dimmed out and we're getting red squiggly line under the package reference here because of course we're not using the validator just yet but we will do in just a bit so we'll add the other import references as we develop the register user endpoint handler function so let's create the basic structure for the register user endpoint handler function like this so let's type funk register user we want this to be exportable so the first letter in the name of this function is capitalized the R and then let's include open and close round brackets and then jin whoops dot handler funk to declare this as a handler funk an endpoint handler function like that a jinonic endpoint handler function if you like and then let's return an anonymous function that we'll implement where we'll implement our logic for this handler function and we want to include the C parameter which is of type jin.context which will help us implement the ginonic functionality so for creating requests returning responses etc let's declare a variable named user of type models do user which is of course the user strct that we implemented that we just implemented within the user_model.go go file this user strct here and the next step is to create an if condition if C that's the generic context dot should bind shouldbind JSON is the one we want should bind JSON so let's select that there we include the amperand user because we want to reference a particular address in memory so where that reference it's a pointer to memory rather than than the actual data at at this point so we need to pass a reference to a memory address rather than the actual data that's what the amperand means preceding the user variable name there and then colon and let's include an object there and an if condition so if it's not if the variable is not equal to null that means an error occurred during the binding process so let's send back an appropriate message a HTTP status of status bad request to the client and then an appropriate error message and we'll use Jin to help us send back oops nil should have one L here and we'll use Jin to send back well-formed JSON to the client so error and then colon and then let's send back an appropriate message invalid input data so if it can't bind successfully here it's invalid data and then of course we don't want code to continue execution because the handler function would have failed at this point so we include the return keyword which makes sure that the code stops at this point once the relevant error message has been successfully returned to the client okay and then let's create a new validator so to do that we go validate and then colon equals the assignment operator and it also declares a variable of the relevant type at the same time that's why we have a colon preceding the equals there so we go validator dot new like this and then we do we check we check an error object if the validation so validate strruct if the valid if this validate code fails it will return an error so we need to include an object here and then user our variable name so this code will return an error value and assign that error value to this variable here and then we can continue the code by adding a semicolon here and checking the for a value to see if an error has actually occurred during this validation process okay so if it has we want to output an appropriate error message to the client so we want the status to be HTTP dot status so it would be a bad request if the inputed if the client user data passed to this handler function fails validation we deem this as a HTTP bad request so we send that back to the client as well as an appropriate error message so jin.h so we can include well-formed JSON so we can send well-formed JSON back to the client if indeed the validation fails let's include the main error message which will be validation failed and then we also want to include the details of why the validation failed so details like this and then we can include this code error and then the error object will have an error property that we can leverage here in our code so it's an error method it's an error function so it will return an appropriate message which will contain the details of why the validation failed so we can send that back to the client and then we include the return keyword here to end execution of the function because the function would have the handler function would have failed at this point here okay excellent and now a very important part of the code before we insert because if it's past validation we want to insert it into the database into the users collection but it will contain the user's chosen password the user variable will contain the user's chosen password and we don't want that password to be saved as is to the user's collection within our MongoDB database because that's a security risk so what we want to do is hash the password so we're going to use a particular package to hash that password so let's install the relevant package but we first need to make sure that we're in the root of our application where our gomod file exists so we go cd server and then mad stream movies server like that check that that's correct yep and then let's press the enter key so we're in the appropriate directory and now we can use the go get command to install the relevant package that we'll leverage in order to hash our password before the user data is saved to our database so let's go let's uh use the goget command go get golang.org org/x/crypto slashb crypt corrupt crypt not corrupt okay so bcryptrypt so golang.org/x/crypto org/x/crypto/bcrypt press the enter key okay and it's doing its thing it's installing the relevant package downloading go.org xc crypto the appropriate version excellent patience is a virtue I've been told so we're getting there and then we'll be able to you leverage the functionality within this package to encrypt the password which has been passed through within the user data passed in from the client okay great so let's clear the screen and now let's create a function called hash password that will hash our password that will encrypt our password before it is saved to the users collection so funk hash pass word this is the name of our method and we're going to pass in whatever is passed in through the client from the client to the server and it's a string value and this returns a string and an error object so that's what we're declaring here here's our return types a string and an object let's open the curly braces like that and let's go hash password so this is one of the return values and these are the return values from our B crypt dot generate from password and then we want to include an array of bytes so we're type casting the password into an array of bytes like this password which gets passed in as an argument here and then we tell it what we want to it to do so b cryptdefault cost okay and then if so we're checking the error if an error occurred during this hashing process we want to handle it and let's return so we're not we're going to return an empty string here for this type here if an error occurs as well as the error object itself there and then if all is successful we want to type cast the hash password into a string so we can do that with this code string hash password and the error that we'll return this object here should be null so we can just return null like that and that's our hash password code written it's as simple as that so we can now use that within our register user code to hash our password before we save the relevant user data to the users collection so let's go back down here and let's implement the relevant code so we go hashed whoops hashed password colon equals 2 and then we can call our function that we've just written and pass in user password like that and that's the password that the user of course has passed in to our server code here that needs to be hashed before the user data is saved to our database and now let's create the context that helps us clear up resources for this function resources created to interact with our MongoDB database so we're familiar with this code now because we've implemented quite a lot already in this course so with timeout and then context dot background and then 100 times so it's going to time out the code will time out in 100 seconds so if something goes wrong the resources will still be cleared up in one and of course we need to include the defer cancel code here like this okay brilliant and we haven't included the time package so we need to do that let's include the time package here time and it's seems to have automatically added context and net/http automatically so make sure you also include these imports within the import section here great so that's excellent and then the next thing we want to do is make sure that the email address passed in for the particular user for the specific user is unique i.e has not been saved to the users's collection so everybody's so members of the magic stream website must have a unique email address if they want to register with our system so we need to check that we need to validate that so to do that we firstly need to include the users collection so if you'll recall we've got this code here which which opens the relevant collection so this is the movies collection in this particular case but we want to open the users collection within our MongoDB database so that we can save the relevant user data to the users collection so I'm just going to make a copy of that because the code is very similar and then just change the relevant bits okay so let's do that just outside of any particular function we can include that code and we'll go user collection like that mongodb Mongo collection so we actually need to include our MongoDB driver for Go packages here in order to make use of this code so let's replace that with users because we want to reference the users collection here and we need to now import the relevant packages here so we can actually copy the relevant code here to do that so we've already included the relevant importation code here within the movie_controller.go file so let's just copy that code go back to the user controller.go code and include those package references the import the relevant importation code here right and you can see when I saved that it actually automatically imported our database package and that's the database package that we created based on the code that we written here we have a reference to database here so when I saved it Visual Studio Code actually did the automatically included the relevant importation code within the import section here which includes our database package so if we look at our database package here now all the exportable code within this package can be leveraged from within our user_controller code and that's what we're doing here that's what we're doing here so now that we've got the user collection variable we can use that to interact with the users collection within our MongoDB database so we want to count how many documents contain the email that has been passed in from the client to server so passed in for the relevant users details from the client to our server code here so we're going to now leverage the users collection variable here we don't want replace one we want to leverage this method count documents ctx so we include our context here so that the resources will be managed appropriately based on the code that we've included here in this section then we can filter our data using this code so bon like that dot m like that and I think we need to include another package here yeah we need to include another package so let's go back let's go to the movie_controller go code here oh no we've already Okay so we have included the BSON package no we haven't oh okay so for some reason it Okay so when I saved the file it actually removed that package automatically that I had included here because we're not leveraging any of its function we weren't leveraging any of its functionality at that point so I'm going to just copy that package again to the user_controller file here like that and let's save that and now that we're using it it won't delete it okay so we can bon oops bon m and we want to use bon.mm to filter our result by the email address passed in from client to server so we go user email here right okay so we need to as always check the error if not equal to null we need to handle that so c.json JSON http and this will be a status 500 internal server error if this code has failed we want to throw a status internal server error 500 HTTP error and send that to the client with some well-formed JSON giving a bit of context to the error scenario okay so error and then user already exists so they got to choose a unique email address okay and then let's include the return keyword here so that it halts execution of this handler function brilliant and then the next thing to do is just Oops no that's the wrong error message sorry so this error message is failed to check existing user so sorry wrong error message so failed failed to check existing user here so if it fails at this point it's actually failed to check the email so the code itself has failed it hasn't actually performed the it has it's been unable to to perform the appropriate count of email addresses currently saved to the users's collection so this is an appropriate error message okay so now we'll check to see if the count is greater than zero and so if the count is greater than zero we cannot add the user data passed in because the user's email address already exists within the users collection within our MongoDB database so let's now include an appropriate status which in this particular case will be HTTP status conflict because the email address conflicts with an already existing email address in the users collection jin.h H and now this error will be user all ready exists like that okay so that's where we'll include the user already exists error if the count is greater than zero meaning that that email address already exists in the database and then of course we want to return we want to include the return keyword here so that the code holds because this handler function would have failed at that point great okay and then I'm going to include a unique user ID and I'm going to say user ID equals to bson dot we're going to leverage the new object new object ID.hex hex method to just create a unique user ID for the user okay great and then usercreate at will equal to time dotn now so we're just going to include what the time is now for this particular field at this point at the point where the user data is being added to the users collection and the user do update at property or uh field should also be set to whoops not that copy that time.now great and then lastly we want to add it add the user data if it code gets to this point we're in good shape and we want to use the user collection variable and the insert one method to insert the user into the database okay brilliant and then we need to check the error object if error is not equal to null yeah oops if is not equal to null let's handle the relevant error so c.json http status in internal server error and let's include let's use jin gonic to include an appropriate error message so it's going to be error colon and then failed to create user like that and then of course we include the return keyword so that execution holds at this point and you can see there's a red squiggly line under result because we're not yet using the variable and at this point we can return the result to the user to the client calling code like this json http status created so it'll be an HTTP 2011 status that we'll return to the client as well as the result there and you can see the red squiggly line has disappeared okay and one thing we didn't do is we didn't assign the hashed password we hashed our password but we didn't assign it to the actual relevant user password field so we need to do this equals to hashed password so that was nearly a mistake there okay so it's got a warning let's just check this value of error is never used ah okay so we're not checking the error at this point yeah okay so we need to check the error okay so if error not equal to null close brackets and we can that would be a status internal server error so let's just take make a copy of that and paste that here error unable to hash password and the warning squiggly lines the yellow squiggly line disappears and all looks good with our function and that's it we're in good shape so next we're going to test our function through Postman great so firstly let's create a route for our register user handler function let's go to the main.go file here let's just create a duplicate of this copy paste and let's map our register endpoint so for slashregister is our endpoint here and we want to map it to the function we've just created which is register user like this there we go excellent so we're now ready to test our register user functionality the functionality we just created here which essentially saves a user's registered details to the users collection within our MongoDB database and we're doing lots of other things here we're hashing the password before the users details get saved to the users collection and we're also checking for any conflicts in terms of the user's email each user must have a unique email great so now we are ready to test our register user functionality through Postman so let's launch Postman here i've got Postman loaded here already let's set the drop-down list here to post because we're about to test a post request let's enter the URL so our end point is for slashregister we've got the base address here already in place and then we just need to include for/register which is the end point maps to our register user handler function and then we just select the body option here and raw and we need to populate this text box with the relevant JSON to register a new user now I've actually already created some sample data that we can use to test our functionality so let's go to GitHub specifically my GitHub repository github and then we want to go to so this would be github.com/gavlon digital and then we want to go gavlon digital magic stream here and then click on the magic stream seed data folder here and then I have prepared this data here so add test user doc and all we want to do here is copy this code oh sorry this data here into our text box here and then to test our functionality we simply press the send button let's do that and of course that's not going to work because I haven't run the code okay so let's go back there and let's run the code so let's go to our terminal window here and type go run dot now the code is running excellent listening on port 8080 and now let's try that again so let's go to Postman press the send button 201 created that looks good ah okay we have a problem so we got a series of zeros here again and why hasn't that worked i thought we had taken care of this bug so let's go and look at our user strct and see if there's anything funny there okay user model and if we look here there's a gap between this comma and omit empty remember this was the problem last time when we tried to save a movie to the movies collection it did the same thing it created a series of zeros for the ID and we want this to be a unique ID field so what's actually happening here is because the formatting here of these tags is very temperamental it's caused a problem because we've got a gap between this comma here and the omit empty tag so let's just remove that gap um save that i'm just going to cancel this by pressing Ctrl C let's clear the screen let's run the code again go run dot press the enter key and now we're running it again and hopefully this is will resolve our issue and we'll the result will contain a unique identifier for the user that we are saving to the database but firstly let's delete the data that was created cuz it did actually create a document within the user collection you'll see here reload data but this object ID is incorrect this ID that uniquely identifies the user is a series of zeros and that's not what we want so let's delete this record by pressing delete there and then confirming the deletion you see here document flagged for deletion and then let's confirm the deletion by pressing the delete button here and now that record's gone so if we reload data we can't see that record anymore for Craig Denton great so let's go back to Postman you can see I'm still running the code here i haven't stopped the code from running and let's try that again and we're all set up here our data doesn't change that data is fine and see the password here password one exclamation mark now this password I'm using for all the users just to make things simple so for all the users you'll be able to log on with this password here but you'll see when it goes to the database it will be hashed i.e encrypted as a security measure you shouldn't save the password as is to the database so we want this to be encrypted and um we've done that through this encryption code here we're actually hashing the password we created a hash password function here using the b-rypt package here to hash the password so when we look at the database we want to see that the password has been hashed once we've saved the relevant data to the database of course so let's go back to Postman and all we need to do we're all set up here we've got the post drop-down set we've got the correct path to the endpoint the register endpoint and the data we copied from GitHub and we're ready to test so let's press the send button here 201 created that looks good but is our ID unique now and it is so that did fix the problem and of course the problem was this silly little gap was caus a gap between a comma here and omit empty tag was causing the problem it's a bit temperamental in terms of formatting of these tags so we got to be mindful of that with this logic here okay brilliant so let's take a look at the at compass to see if our data has been created our resource has been created within the users collection so let's reload the data by going view reload data and there it is and it's got a unique ID here which is fantastic great so we're in good shape that's worked one more thing I'd like to test is if we now try to add that same record we should get a conflict with the email because each user within the collection within the users collection must have a unique email address and of course we've just seen that Craig Denton with this email address has been saved to the users's collection as it were so if we send that same data to this endpoint we should witness a conflict here so it should send back a HTTP conflict response HTTP response to us uh HTTP status response and it and that will result in the data not being duplicated within the users's collection so we don't want to save another document that contains an email address that already exists in the users's collection so a conflict should occur when we hit the send button here because this data contains an email address that already exists within the users's collection so let's try that let's press the send button 409 conflict brilliant so that's working user already exists as our error message sent in JSON format to us to the client code so you can see that here 409 post it's resulted in a 409 conflict brilliant so the data won't be duplicated if let's just confirm that by going to compass here and we should only have one record for Craig Denton excellent so our code that handles the conflict has executed successfully and let's look over here register user you can see here the code for that count documents filtering on email if the count is greater than zero error user already exists brilliant so now we're ready to write the code to write the login code on the server side excellent so now that we've registered a user successfully in the system let's create the user login functionality so let's first open the user_model.go go file here and let's create a model for storing the login data that will be passed from client to server once the user has submitted the user's login details in order to authenticate the user's security details with the magic stream website so let's create a strct named user login here we do that by typing type user login is the name of our strruct and then we need to include the strruct keyword here open our curly braces and we want an email field like this as string and then we can include the relevant tags for our email field so let's open back to characters like this include the JSON keyword here and then the name of the field that will appear within the JSON data and then validate let's include some basic validation here so it's required field is required email must be in a valid email format and then let's include the password field like this string open back tick characters so the field will appear as password within the JSON data validate so it's obviously required min equals to six so must contain a minimum of six characters the password in order for it to be valid okay okay so let's create the user response strruct which will be a subset it's a DTO and it's a contains a subset of the fields contained within the user strct so let's type type the name of the strct is user response like that and then of course the strct keyword open and close curly braces like that this will not actually map to a BSON field so I'm going to remove that so the BSON code doesn't need to be included here because it's doesn't map directly it's kind of a subset of the user strct and doesn't map directly to a document or a collection of documents within the MongoDB database so we don't need that bon those that bon tag there but yes we do need the JSON tag to stipulate how we want our JSON data to look okay perfect so right so within the user response let's create a field called user ID as string as string let's open the back tick characters like this and let's include JSON and within quotations user ID like this let's include a first name field like that camel case is what we're using not camel case we're using Pascal case where each word has its first letter capitalized first name each word within the variable name or the field name in this case has its first letter capitalized this is Pascal case okay string and let's open our back to characters and include the JSON tag here and we want our JSON to look like this first underscore whoops name let's just create a copy of this paste it just below and this will be last last name okay and let's alter this tag to be last name and then let's include a field named email as string let's open back to characters here json colon email like that okay and then roll string open back to characters JSON we want the JSON to look like this roll all in lower case and then favorite genres okay and this is an array of the genre type this is our userdefined strruct that we created earlier in the course okay and let's include the relevant JSON tag here and we want it to look like this within the we want this field to be named like this within the JSON data sent back to the client this model is a DTO which stands for data transfer object so DTO stands for data transfer object it's a design pattern used to transfer data between software application layers such as between a backend and front end or between services in a distributed system here are some key characteristics of a DTO it holds data only it typically contains fields properties getters and setters and no business logic it is a flat structure usually simpler and flatter than domain models it is serializable designed to be easily serialized to JSON XML etc for transmission over the network some common use cases are as follows api responses returning data to clients without exposing the internal domain model so in this case the internal domain model would be user and this is our DTO user response data aggregation combining data from multiple sources into a single object input validation accepting structured input from external sources for example form submissions so by using a DTO we're only exposing the data that needs to be exposed to the client great so let's write the code for our login user HTTP endpoint handler function so let's open the user_controller file here and let's create the infrastructure for our handler function so let's type funk login user like this then jin dot handler funk so we're using gingonic to establish this function as a handler function an endpoint function handler and we're leveraging gingonic the jinggonic web framework for this purpose and then let's return the logic for this particular handler function so we do that by return funk and let's open brackets c and then this is of type star jin context like this let's open and close curly brackets for our anonymous function we're now going to implement the anonymous function that we're returning from the parent login user function we need to make sure that whoops we haven't got open brackets there so we're going to open that bracket there and close it down here okay and then let's create a variable that's named user login and this is now in camel case where user has a lowerase letter starts with a lowerase letter and the next word login starts with a uppercase letter so this is camel case and we're defining it as type models dot user login like this and of course user login we've just defined within our user_model.go file it's a userdefined type that we've created okay great so we've created our variable user login and let's bind the data sent in by the client which should only be the user's email address and password to the variable we just created named user login okay so we go if so we're going to check the return of a method provided to us by the context of jinggonic which is called should bind whoops bind JSON so we're checking the error of this binding functionality and of course the amperand user login points to a particular address in memory rather than containing the data itself this refers to a pointer in memory and that's denoted by this amperand character preceding the variable name and let's include a semicolon and then an if condition if error we're just checking if the return value is null if it's not null we need to handle the error should bind JSON sorry this has to be capitalized json the red squiggly line's gone away okay so C.JSON let's handle the exception http dot we want to send back a status of bad request here comma jin.h error and the message we'll send back is input oh sorry invalid input data like that okay okay and then we need to include the return keyword so that execution ends at this point here the function would have failed at this point here if there's a problem with binding the data passed in from the client which is the user's credentials the email address the user's unique email address and the user's password is passed in here and we're binding that data to our user login strct here the variable defined as the user login strct here and if it fails we're handling the exception there so the next step is to find the user within our users collection within our MongoDB database based on the user's credentials let's define a variable named found user but I'm actually going to set up the context for our query where we're finding the user within the users collection so we've done we've written this code many times and this is just for handling resources so even if our query fails for example within this function that the relevant resources are freed up with timeout like this open brackets there and let's include context dot background and let's so we want all the resources to be cleared up in 100 seconds so time dot second like this and then we also need to defer cancel with this code here so let's define a variable named found user as the models do user type so we type var found user models do user like this okay and then let's include our query to our MongoDB database so we can do that like this so assignment operator users collection oops we don't want that so it's users collection or was it user collection what have I defined it up here as user collection okay so it's not users collection and it's user collection user collection dot find one we want to find one document within our user collection let's include the context here and then let's filter our query m on the email the the user's unique email which is being passed in from the client because the user is logging on in this scenario so while the user so when the user logs on and presses the login button the user's email address and password are sent to this function here and we are filtering on the user's unique email data in order to find that user within the MongoDB database within the users collection so find one email and then user login we've bound email we've bound the what's inputed from the client the email and the password to this memory address here we've bound to this variable so it's pointing at a particular memory address with the relevant client data and we're now querying on the email address to find the data and if it's found we want to decode that into found user our found user strct like that our found user variable email of course the comma is the issue there so let's replace the comma with the colon okay and then let's check the error and handle it if an error indeed exists so error so this is the error caused potentially caused by the find one functionality here where we're trying to find the relevant user within the users collection so if error null if error is not equal to null whoops open our curly braces and let's handle that particular exception so we want to send back HTTP dot status unauthorized okay we can't authorize this user because we can't find the user within the database and then let's use jin.h to hand to send back an appropriate error in JSON format to the user invalid email or password and then let's include the return keyword here so that execution is halted at this point so the function would have failed if this method fails it's pointless carrying on with the execution of this code so we handling the exception and sending back an appropriate error message to the client invalid email or password great so the next step is to compare the password passed in from the user during the login process to the user's saved password within the MongoDB database but of course the user's password is stored in the database as encrypted data so we can actually see that if we go to compass let's look at our users collection quickly before we carry on writing the code let's look at our users collection quickly so let's go to Magic Stream Movies if we go to the users collection here you can see that the password is hashed so this is pretty meaningless to a human you the user can't log in with this password it's deliberately being encrypted to increase security so nobody can steal the password if they somehow get access to the back end to the data store here okay but we can use a particular package the brypt package to compare the hashed password with the password passed in and let's look i think we've already imported the relevant package let's have a look up here and here it is brypt so we don't need to install anything or import anything it's already done for us we've used it before so let's write the password comparison code so equals to b crypt that's the package we're using for this comparison code compare hash and password so it's doing it for us so we don't have to do anything write anything fancy because it's already encapsulated within this functionality here within the package that we're using here so we want to type cast found user password into a bite array here comma and then this is the use this is what's being passed in this is the password that has been passed in from the client during the login process dot password and we need to also convert this into a byte array and we're using this code to do that so we're comparing like for like here essentially compare hash and password and that takes care of all the details of comparing a hashed password to a to a password that hasn't been hashed so it's been done for us using the brypt package here okay and then let's write the code to check the error if something went wrong during this hash comparison process nil let's go here so let's type c.json and let's pass back http unauthorized if the password doesn't match up this user cannot be authenticated and we sending back an appropriate message to the user as well as a HTTP status of unauthorized so error like that and it's the same error message essentially as before invalid email or password like that and of course we need to include the return keyword here so that execution halts at this point because there's no point in executing any code that might exist after this code here because an error occurred during the comparison process is the compare comparison of the client's password that the client has passed in during the login process to the password saved to the relevant users document within the users collection so it will fail at this point and no further code will execute if no error occurred and the user's password matches the password saved for the user in the user's collection within the MongoDB database the user is authenticated and our code must now pass back to the client an access token so what we are going to do now is create code for generating an access token and a refresh token which are generated once the user has been authenticated so the access token is analogous to a key that can open the doors to protected resources so this is great for when you have an application with disparate loosely coupled components that need to securely communicate with each other in our application here we have two disparit loosely coupled components a client web application front end which we'll create later using React and a serverside web API component this is the component that we are currently creating using go engine gonic so when the user logs in and is authenticated an access token is generated on the server this is functionality we are about to create the token generating functionality if you like the tokens are then passed to the client the client code can then send the access token to the server along with the relevant HTTP messages in subsequent interactions and won't need to provide the user's credentials i.e email and password with every interaction with the server because the access token can now be passed from client to server which means through the access token the server code will know as it were what authorization privileges the relevant client has in relation to the relevant endpoints some of which will be protected so we are going to write the code for generating the access tokens within a separate file so let's create a file within the utils folder named token_util.go like this okay so we've got the utils folder here let's right click go new file and we are calling this token util like this go great so at the top of the file let's declare the name of this package which will be util so we just type package utils like that let's create an import section like this so import and we've done all this before and then we just open round brackets like this and we can include our imports here let's import the MongoDB driver for Go packages here like this so we're going to we've already imported these packages before so let's just go to the movie_controller file and just copy these packages these package references where we're importing these MongoDB for Go driver related packages here let's copy that to our clipboards let's go to token util here and paste the relevant importation code here excellent okay okay and we've got red squiggly lines under here because we're not currently using any of the functionality associated with these to these packages here we want to also import a package that references our database functionality here where we're connecting to the relevant database and opening a connection to the relevant collections here with this reusable code so we want to include a reference or we want to import this package into our token util file here or our utils package rather okay so let's press enter here and then we're already importing the relevant database related package here so let's just copy this let's go back to our token utils file and paste it here great and to generate the tokens we need this package github.com/golang-jwt/jwt/v5 okay so we actually need to install this package so firstly let's navigate to where the go.od file is so cd server slash mad whoops yep magic stream movies server like this and then let's run the relevant goget command so go get GitHub here's the path to the package that we want to install com/go and we need this particular package JWT and we need this particular package for generating our access tokens our JWT or Jot tokens jwt slashv5 five and let's press the enter key that looks good and hopefully that installation process will go as expected and then we can proceed with generating our tokens or writing the code for generating our relevant tokens and that looks great excellent so let's clear the screen here and let's import that package now so to import the package we just include this line of code here github.com/golangjwt/jwt slashv5 like that okay so we're creating a model here and we're going to call this sign details this is all part of generating our token so just bear with me here and follow along please so we got email as string whoops first name string la whoops last name string and these details will all be encoded within the relevant JWT this is why we're creating the strct then we got RO so this can be admin or user so let's define it as string is the enter key user ID as string and then lastly JWT dot registered claims so we're including a strruct within a structure basically this is the JWT strct within this package here so actually we need to alias this here so let's include JWT here just for readability purposes so we can reference that within our code here so JWT dotregistered claims and we're basically including another strct within a strct and that strruct resides within this package here so our token is going to be generated from this information email first name last name RO and user ID and then registered claims which is actually a strct that we're including within our signed details strct here jwt.registered claims is a strct provided by the github.com/golangjwt/jwt/v5 package it contains a set of standard claims defined by the JWT specification RFC 5719 and these fields include the issuer which is who issued the token the subject the subject of the token usually a user ID audience the intended recipients of the token expires at when the token should expire not before when the token becomes valid issued at when the token was issued id unique identifier for the token can be used to prevent replay attacks okay so let's create code to read a secret from an environment variable so to do that let's go to the env file and include an appropriate environment variable and our secret key will just be secret underscore key equals to and we're just going to call this your secret key you obviously want to include something a little bit more a little bit better than this but for now this is just a placeholder to indicate where we are storing a particular secret key which will be used to encode our token to create our token as part of the token creation process and you'll see how we do that in just a bit okay so we've included our secret key here let's go back to our token_util.go file here so let's create a variable called secret key as string and set it to our secret key which has now been set within our env file okay so we go var C whit key as string equals to and then we need to include the OS package dot get env and then the environment variable that we want to read is secret key so we can just copy that go back to our token util.go file and copy it there so we're now reading the secret_key environment variable and saving it in memory within this variable here which has been defined as string now we need to import the OS package so let's go up here and within quotations include OS like that okay excellent great so I've noticed it does this annoying thing when you save your file and some of these packages are not being used that it actually just uh Visual Studio just deletes them automatically so if I go save it deletes them because they're not being used so um that can be a little bit annoying because we haven't had a chance to use the functionality within those packages so I'm just going to undo that and just just be aware of of the fact that it will remove those if they're not being used and you go Ctrl S to save the file okay so I want to keep those packages there and we're going to start using the functionality within these packages so those red squiggly lines will disappear great so let's create the generate all tokens function which contains the parameters email as string first name as string last name as string ro as string user ID as string and returns signed token as string signed refresh token as string and as error so let's create that function definition so let's type funk like this generate all token so it'll be an access token and a refresh token we'll discuss what a refresh token is a little bit later but for now let's just create the definition for our function email so comma first name and you can see all these parameters are established using camelc case last name ro user ID okay and we can include string so we only have to define it for the last parameter and all of these parameters are automatically defined as string because we've defined the last one as string and we haven't included types for the preceding parameters here within our function definition and then open brackets we go string string error so we're returning two strings here and an error object from this function let's open curly braces like that press enter like that let's create a claims variable and assign it to the science details structure with its field set to the appropriate values passed in to the generate all tokens function so let's go claims so we're defining a claims variable using the assignment operator like this which both defines the variable as the relevant type so it infers the type as well as it's also doubles as an assignment operator so let's include what we're assigning this variable to and that would be amperand and we know the amperand is refers to a memory address signed details like that open the curly braces like this email and then let's assign that email field to the email value passed as an argument to this function okay and then let's go first name like this colon and then the first name parameter value do the same for last name last name and then we want to assign the role like this and then user ID oops user ID and that should not be capitalized ID like that and then registered claims and we can actually define the registered claims that we want to include registered claims so this is a strct and we can define what our registered claims should be here so we've defined the strct we want to include and we're going to include the relevant fields that we want to include within our claims here so as you can see what's happening with this token is we are encoding the users specific details into the actual token okay so the issuer is the app that's issuing the token so we go magic stream it's not issuers it's issuer like that and then issue issued at like this colon then let's use our alias for our package that we recently imported new numeric date time dot now okay okay and let's just make sure that time time should in fact be imported here so let's include the time package here before it complains so we got time now and then comma here and then let's go expires at and include the relevant expires at value like this time do now and let's add 24 hours and this is when when it will expire so it expires within 24 hours we might change this a little bit later but for now let's just say 24 multiplied by time dot hour so it expires in 24 hours this particular token okay and that looks good excellent and the underline there is because we're not yet using it okay and then let's create a variable under this claims variable let's create a variable called token token assignment operator JWT i know this is very involved but just bear with me this is pretty much a standard way to generate the tokens it'll be worth it in the end okay so new with claims and we're now passing our claims which just contains relevant details about the user sign and this is the algorithm that we're going to be signing our token with which is uh ES256 here and we actually want HS I think so let's go sign so HS 256 so this is the algorithm that we're using to sign the claims and then let's include the claims variable there so this is our token generation code and then we want a variable called signed token like this cuz this method is going to return two values then our assignment and declaration operator and we go token dot sign string so now we're going to sign what we've got in the claims with our secret key byte so we are converting the secret key from a string into a byte array for this method to work so sign string accepts a byte array so we must convert the secret key string into a byte array and that's what we're doing here great so we're getting somewhere and then and then and then we do the usual go thing we check for errors so not equal to null and this is the thing I had to get used to coming from C to Go is the way errors are checked it's very procedural rather than using try catch code like you would in JavaScript C++ or C okay so let's write return and then okay so if an error occurred we're we're returning nil nothing an empty string for the first one so the token will have nothing in it there and the refresh token will also be set to an empty string so an empty string for the token an empty string for refresh token and then the error object which will be useful for the client code so the client code calling this function will be able to handle the relevant error sent back in the case where an error occurs great so then for the refresh token is very similar okay okay so we're first just going to go to the N file here and we're going to do a similar thing here so we're just going to create a key for the refresh token secret reresh key like this your refresh key of course you'll probably want to include something a little bit better than this but for now it will serve our purposes and let's go back to this file here and then all we what we can do here now is just duplicate this code for the refresh token okay so let's do that so actually we can just duplicate this code i know this isn't very good but just for now to get to the next step so we can just generate our tokens we can always come back and clean up this code a little bit later so let's call this refresh claims it's obviously complaining because we got duplicate variable names now because we've duplicated the code to generate the token we're generating the refresh token now and we're duplicating the code that we created for generating the access token so we just need to change these variable names so that all these red squiggly lines will go away it's obviously complaining about duplicate variable names okay so then we can just change that there we can call this refresh token and use camel case for our refresh token variable and then signed refresh token here token and then of course our secret key which we need to read here now so we've called it something different so go secret re refresh key let's just double check what we called it there see yep secret refresh key and we can just paste that like that there okay token refresh token that's these red squiggly lines are just because we we're not currently using the variable and same with that variable there okay all of that looks pretty good so that needs to change to refresh token refresh claims signed refresh token okay we're checking the error there to make sure no errors have occurred during the refresh token generation process if you like okay so then um checking the error so if no errors have occurred the only thing left to do here is return the token refresh token and hopefully a null for the error but if there's an error the tokens will contain an empty string and the error will contain a value but we're hoping that the tokens the signed tokens the access token the signed access token and the signed refresh token do contain values in which case the error will be null and no error has occurred signed refresh token null there we go whil and now do we have any red squiggly lines no and so it's cleared out some of our references here because I saved it so that those were the MongoDB for Go driver related packages let's cleared those out cuz I haven't yet used those but we will be using those because we want to save these tokens ultimately save these tokens to the database next to the relevant user within the relevant user document within the users collection okay so let's actually write the functionality for that now because we want to include it within this file so we're going to write the the actual database related functionality now so I'm just going to bring back those package references it's a bit annoying that it just deletes them um automatically uh because we're currently not using them when you save the file but okay great and see I just saved it and it went away so undo that we got our packages here and I just want to open and it's also deleted the database reference too which we need so let's go here database so we just need this database importation code here and let's copy that here and then let's open a connection to the users collection actually we've got the code here already um here so let's copy this code here let's go to token_youutos.go paste it there so now we are using our database package so let's create the function to save the token so we we've generated the tokens in this code here we're going to be calling that from the relevant controller code we'll generate the tokens and then we want to update the users collection with the tokens the relevant user document within the users collection so let's create our function we're going to call this function update all tokens so let's type funk update all tokens like this open brackets and we can go user ID like that token like this comma refresh whoops refresh token like that and then just define this last one as string this last parameter string which by default will automatically define these parameters also as string and then we just want to return an error object so go as error like this open our curly brackets and let's write the logic for this so firstly I'm just going to create the usual resource clearing code if you like the housekeeping code that when time when the timeout is reached it clears up all the resources created when using the mongo forgo driver related code so width time out then we want context oops not tudo dot background we of course need the context package imported which which is it's done that automatically for me there okay and then let's establish the timeout and let's just use our usual 100 times time dot second for the timeout okay so it'll time out in 100 seconds and then we need to include the defer cancel code here okay brilliant okay and then date at we don't need it to return the second value here so we just include a placeholder here which is an underscore character and then we want to go time dot pars time dot r and this is just the standard rf c 3339 like that let's go time dot now time dotn now dotformat and then we want it in a particular standard format like this okay now we're going to create a variable that assigns the relevant values to the relevant fields within our relevant user document so update data let's define those fields and we can go bson dot oops bson dot m like this and then use the set command like this which is dollar symbol set like that colon and then another bson m like this and then we can assign the relevant values to the database fields like this token tokens being passed in as a parameter value here and then of course refresh token like this okay refresh token like that and then update at which we've established here and this is really for auditing purposes so when a document is updated in the database we have the timestamp next to it when it was updated and we can go updated at like that there so we've defined which fields will be updated in the database and with which values okay and we need a comma here great so we're using this variable to interact with the relevant database collection which is our users collection so let's include underscore we don't need a value returned for the first return value but we want the error returned if an error occurs during this update process let's go update one like this so we're calling the update one method using the MongoDB forgo driver related functionality so let's include our context here and then user I whoops user ID field we're filtering this particular update we're filtering on the user's ID so we're targeting a specific document within the users collection based on the user's ID great and then the update data variable that contains how we want our document updated the fields and the values for those fields so this code here this set command will set the relevant fields in our document with the relevant values and we're using the user ID to target a specific document so it's a specific document for a specific user and we're updating that particular users tokens the access token and the refresh token so that's what this update one functionality is doing here and then the usual go thing where we check the error hopefully it'll be null but if it's not we'll just return the error to the client calling code and then if it gets this far we return null for the error like that and that should be good enough to update our database okay excellent so we've now written the functionality for updating the relevant document filtered on the relevant users ID and we are ready to create our calling code within the user controller so let's go back to the user controller and back to the login user function and we can proceed from here so we can now call our generate all tokens method from within our login user function to generate the relevant tokens so let's firstly do that so to do that create a variable called token and then refresh token like this and then the operator the assignment operator u utils dot and I assume it's already imported utils for us here okay no we haven't yet imported the utils package so to do that let's copy that there enter utils like that there great so now we should be able to go utils dot generate all tokens like this and then we can pass in the found users relevant fields like this so found user email so these are the fields that are used for creating the relevant tokens so then found user dot first name found oops found user.loast name found user dot roll like this and then found user do user ID and that's the last parameter we need to include there and there's a red squiggly line because we're not currently accounting for the third return value which is the error if one has occurred okay perfect so that looks good and then we've got to go through the motions here and check the error so if not equal to null so that obviously means an error has occurred we can handle that using the gingonic context so go there http status internal server error and then we can return wellformed JSON using jin.h like this go error and then fail to generate tokens so hopefully the code doesn't reach this point and the tokens are generated as expected and then we use the return keyword at this point because we don't want the code to continue executing okay and then at this point in in the code we actually want to update the tokens within the relevant user document so to do that we go utils and we wrote update all tokens that was the last function that we wrote within the utils package so update all tokens we need to pass in the user ID so found user user ID like that and then token and refresh token token and refresh token like that so that the relevant tokens are saved to the relevant user document within the users collection okay and then once the user's logged in we want to pass the user response now this is the DTO we created earlier so if we go to models here user model we've got this user response which with which is our DTO and what I didn't include in this DTO are the tokens at this point in the application development process we're going to return the tokens but in subsequent iterations of the function we're actually going to save the tokens to what's known as HTTP only cookies but for now we'll return the tokens to the client through the user response DTO and I'll explain more about why we are doing this for security reasons a little bit later in the course so let's include token here as string and we want the JSON to be straightforward just JSON colon and token and then we want refresh token which is very similar like this and of course this will be called refresh token and we want the JSON to look like this using snake case with an underscore separating the words and this is of course Pascal case where each word each word's letter is capitalized okay and I think that's looking pretty good and then we can go back to our user controller.go go file and we can return that using the genonic context we can return that user response strruct to the user in JSON format so to do that we type c.json so if the code gets to this point all has been successful we probably should be returning an error at this point and handling it actually so let's just go update all tokens let's go to that definition yeah so we're returning an error there so we need to handle that error first so let's go back to our code here and let's handle the error first so if error not equal to null like this and how should we handle this we should just handle it in a very similar way how we're handling it there failed to update tokens let's say here okay so we're passing an appropriate error and an appropriate HTTP status back to the user here which would be internal server error 500 okay but if the code gets here all is successful and we can return a HTTP status of okay so http dot status okay like that comma models dot and then our user response which is our DTOR here and then open curly braces and user

ID found user dot user ID first name this is what we're returning to the client react code found user dot first name and then last name found user.ast name and then email found user dot dot email and then roll found whoops found user dot dot roll and then favorite genres we also want to return these found user and you guessed it dot favorite genres like that but we also need to return our token so token and we've got our tokens here so token token like that comma refresh token refresh token and that sure that was quite a lot of code but that's how we generate our tokens that's how we authenticate the user and generate our tokens and at the at present we're just going to send them back to the client but later on we'll look at how we can save the tokens to HTTP only cookies because it's a more secure method of passing these tokens between the client and the server excellent all right so we now are ready to test our login functionality but something doesn't look right here looks like we got a warning here oh okay so I forgot to assign the return value from update all tokens to an error variable so let's assign the return value from update all tokens to this error variable here and the warning underline has gone away now and we're good to carry on so now we want to test our login functionality our login user functionality here so to do that let's first run our code ah we got to do we haven't mapped our um we haven't mapped our login user handler function to an appropriate route yet so we've got to do that first so let's go to main.go here let's create a duplicate of this code here just below it and let's call this endpoint login our handler function is of course login user like that and now we have mapped our login endpoint to log the login user handler function and save that and then let's run our code so to do that let's do the usual type go run dot here and it should be listening yep it is on port 8080 let's minimize that and then let's launch Postman we're going to test this through Postman because we're doing a post request here we can easily test get get requests through um our browsers but post requests we should use a tool like Postman makes it a lot easier so firstly we want to make sure that this is set to post and then let's include the appropriate path so the path to our endpoint is login not register in this case so let's adjust this URL here to the appropriate path so for/lo is our endpoint and this is of course the base address of our web API where it's currently running on our local machines and then we want to make sure that we select body here and raw and now we can include JSON data that we want to include within the body of the HTTP message so let's open our curly braces and include the appropriate JSON data let's just quickly go to compass here and look at our data okay let's connect we want to look at our user data because we recently registered a user named Craig Denton so here you can see Craig Denton the user that we recently registered and the tokens are currently set to empty strings so when we log in using Craig Denton's credentials through Postman these tokens should be populated with the appropriate access token for the token field here and the refresh token for the refresh token field here so that's what we want to do here verify through Compass that the login has been successful once we've tested the login user functionality so let's go back to Postman okay and we're all almost set to test our login endpoint let's just go and get the the appropriate data from Compass okay um so let's copy we can copy this email address for Craig Denton so the email address within the users collection uniquely identifies each user and let's type email here colon to include our in fact that should be within quotations so we're including appropriate JSON here and then within quotations let's paste the appropriate email address for Craig Denton and then let's include a comma here and then we want to include Craig Densson's password which is he chose this password because we just want to keep things simple for testing purposes so it's just password it's obviously not the most secure password in the world but for testing purposes this is okay and this looks pretty good let's just verify that the JSON is appropriately formed for what we want to do here so let's go back to our code and let's go to the user model user_model.go file here and look at user login the user login model okay so email is validated as it's got to be validated as a as a properly formatted email address and then we've got required here so email is a required field for the user login strct and password is required and must be must have a minimum of six characters and that looks good so you can see by this JSON tag that our email and password are appropriately named or are appropriately named within Postman let's go back to Postman email password and we've got the appropriate values next to each of these fields here within our JSON and I think we're ready to test the login pretty much so let's do that let's press the send button and see if our login works look at that 200 okay and we've logged in successfully and we can verify that because it's created our access token and our refresh token and of course this is just encrypted data based on the users's claims and various settings that we went through in this section of the course a little bit earlier so let's go to compass to refresh the data we go view reload data and now we can see token refresh token have the appropriate values that has been returned from our end point and you can see that within the result returns to us via HTTP from our login user handler function excellent so now the user if this say was a JavaScript or React client to access protected endpoints these protected endpoints haven't yet been created yet but we will soon be creating these protected endpoints and we can use these tokens we can use this access token to access the endpoint because this verifies the relevant user has logged on and been authenticated successfully so can now access the appropriate protected endpoints excellent okay so I've actually noticed that the access token denoted by the token field value here is exactly the same as the refresh token so something is not quite right with our code so I want to go into the code and just see what could have caused that we want those tokens to be unique so okay let's have a look we want to go to tok to token_util.go and because we duplicated the code for deriving the access token we duplicated it in order to create the refresh token because the code is very similar i suspect that's the reason why we're getting the same token generated so first of all I can see here this should be secret refresh key and let's replace secret key with secret refresh key because we are signing the refresh token and not the access token here so let make sure that's secret refresh key and what else okay here we could up the expiry at field so let's just multiply by 7 whoops multiply by 7 multiply by time dot hour we've got a week before this refresh token expires and we'll look at refresh tokens in more detail a little bit later but I think that looks pretty good let's just test that that now is creating two unique tokens one for the access token and one for the refresh token so I'm just going to run this code go run dot okay great it's listening on port 8080 i'm going to go to Postman now and just run this again okay and we've logged in successfully and let's just compare the refresh token and you can see here that this is now different this is a different value so the refresh token and the access token are different so that looks much better and we did find some slight issues in our code that we've now corrected okay brilliant so we've got the login functionality now sorted let's move on to the authorization code so we want to create our middleware that will authorize the user authorize the client each time the client makes a call to one of our endpoints so some of the endpoints are not protected but others are protected in which case we need to check that the user has been appropriately authenticated so this is the code we're going to create now and we're going to create that in our middleware folder here so let's create a file within our middleware folder and let's name that file or underscore middleware like this.go go it's ago file press the enter key and we're going to make this part of a package called middleware excellent and then let's create the import section open and close round brackets okay okay and let's firstly make sure that we import our utils package so to do that we need to include the root path first which is the root path of where the code will eventually be pushed to on GitHub so it's github.com/gavanlon digital of course this will be whatever your GitHub account is called so mine's Gavinon digital slashmagic stream Movies slashserver slashmagic streammov server i think that's correct let me just double check that so if I go to go domod we can see it here so it's actually magic stream movie server so let's I'm just going to copy that straight from there and you can do the same to make sure it's 100% accurate paste it here and then forward slash utils like that and now we've imported our utils package brilliant so let's create a function called middleware called O middleware so or middleware where like that oops open and close curly braces and this is a jinonic handler function but will not be used in the same way as the other handler functions are used we'll talk about the details of what I mean by this in a bit for now it is important to understand that this function will be used to validate the access token and grant access or prohibit access to protected endpoints jin so we need to hook into the jin gonic functionality so it's a handler function and that means we also need to return an anonymous function like this that accepts an argument of type jin.contextc and we have star jin cont whoops context like this so we're hooking into the Genonic functionality here so we have access to Ginggonics context and you can see that the Genonic package has automatically been imported up here so make sure that this Gingonic package has been imported in order for us to hook into the Genonic functionality here okay let's start writing our logic for the or middleware function so the first thing we actually need to do is create a function to extract the token from the header of the incoming HTTP request so I'm going to create that function in the token_youutils.go file so let's go there and create a function for that purpose so within token_util.go Go let's create a function called get access token like this which accepts a an argument of type jin.context so this is the jinggonic context object and it returns a string and an error so the string will represent the string will be the token and the error will be the error object the typical go error object that we need to return to the client so the client can handle the relevant error saying jin is undefined here okay so we might not be importing the gingonic package here okay so we need to import the gingonic package so let's go back to or middleware copy this where we're importing the gingonic package go to token utils and import it here okay so that red squiggly should go away now vanished yep no red squiggly there excellent but we've got a red squiggly here and what's that saying missing return compiler missing return okay all right okay so that's because we're not returning anything from the function at the moment but let's go through the logic step by step so firstly let's create a variable called token string and camel case like this it's just a local variable for the error object and then our assignment and decl declaration operator c dot request because we're going to be reading the token from the header so C.reest request header.get and then we want the authorization name so these are name value pairs within the header of the HTTP message and we're reading the author authorization um we actually just want to return an orth header object here so let's call this or header so we're returning a reference to this name value pair within the all authorization setting within the header so we need to return an object here firstly press the enter key so if header is empty then let's return nothing for the token so an empty string for the token and then an appropriate error so we go errors the errors object dot new so that should have imported errors here so we need to import errors here firstly for that to work so we've imported errors okay errors dot new and then within brackets we can include an appropriate meth message and this method message can just be authorization header is required okay perfect that looks good okay and then if we get to this line of code here we can create our token string here now token string assignment and declaration operator and then or header now we can read the header and at the same time because the header will contain bearer in the first part of this authorization value we need to extract um the token and leave out the part that has the literal text bearer space in it so we can do that through this code here so len the length of bearer of the literal string bearer space and then include a colon here okay so this should be a little L here len like that and that should work so here we're extracting the token from the authorization setting within the header like this it's a name value pair so authorization is the name so we just want the token part of the authorization setting within the header here so token string assignment operator or header and then we're extracting the token with this line of code here okay great and then we can just check if token string is equal to empty then let's handle the fact that token string is empty so we've we haven't been able to extract the token from the authorization header so it's empty so let's return an empty token so no token and then followed by errors new and let's return an appropriate error to the calling code so we're going to go bearer token is required so it's called a bearer token so bearer token is required will be sent back to the client within the error object else if all has gone well and we've managed to extract a token from the authorization part of the header we can return it to the calling client so return token string and then an error object which will be null so we return null for the error object here okay brilliant so that should extract our tokens appropriately okay so get access token has now been written let's go back to our orth middleware.go file so middleware and then we can call our newly created get access token function like this so token and then assignment operator utils utils dot get access token like that and we pass in the ginonic context to get access token here and as always we need to handle an error if if one has occurred so we go if error is not equal to nil then an error has occurred so let's use the context to pass valid JSON and uh an appropriate HTTP status response to the client so we go status unauthorized in this case because we have received an error trying to extract the token from the header so status oops status unauthorized comma jin.h says pass wellformed JSON back to the client with an appropriate error message error and we'll just pass what's whatever is in the error object like this okay something is a miss and it's of course because I keep putting a comma there and that should be a colon okay and then we can terminate the execution of this function by including the return keyword like this brilliant okay now we want to check for an empty token string and we can do that by going token equals to empty string so if it's an empty string we can return a similar error i'm actually just going to make a duplicate of this and instead of error and then returning the error like that let's return a different message here and we'll go no token provided for this for this error message no to token provided like that we actually need to use the gingonic context object to abort this function and you'll see the significance of this uh a little bit later but so it's the middleware it's the the endpoint that we are calling before any of our publicly exposed endpoints are called by a client so here we just go see abort so it won't continue for example if the user is not properly authenticated it won't continue to call the targeted endpoint because we we are using C.abort here before that endpoint can be called and I think what I'm talking about here will become clearer uh as we progress with this part of the course okay and then we can extract the claims from the token but we need to validate our token first so we need to create a method called validate token so we're going to go back to token_util.go and create another function and this one will be called validate token okay and I'm just going to fast forward the creation of this code for you and then I'll explain what's going on here okay and that looks pretty good let me briefly explain what's going on there so science details is a strct we've defined that embeds or extends jwt.registered claims it's where token claims will be decoded into this uses the jwt.parswithclaims pass with claims function from the github.com/golangjwt/jwt/v5 package it pares the token string into a token object decodes its claims into your claims variable uses a callback function to provide the secret key used to verify the signature you return it as a bite slice which is required this checks that the signed algorithm used is HMAC like HS 256 this is a critical security step attackers can try to spoof tokens using a different algorithm if you don't check this so then we check the expiry date here so this checks whether the expiry at claim a time field is before the current time if the token has expired it returns an error great let's go now back to the calling client code here so we want to call our validation code now colon equals to utils dot validate token and then let's pass in the token here okay then if an error is returned let's handle that case if error is not equal to nil C.JSON http status unauthorized and then gen.h H error and then invalid token okay and then we can abort this by typing abort like this okay we should actually also include a return keyword for each of these error handling cases like this so that the code doesn't continue so we're aborting it from going to the next calling the next end point here but then we want to also abort this function from continuing to execute here okay great and then we've got access to our claims now so we've decoded our claims from the token and we can do something like this which is quite cool we can set the we can use the gingonic context object and set certain values like this and we'll look at the significance of this a bit later claims do user ID so we'll be able to retrieve the user ID from the Genonic context object in subsequent code when it calls when the when the relevant target endpoint is called we'll be able to within that code extract the user ID from the Jonic context because we're setting it here once the user has been authenticated see so the once the token has been validated which means the user has been authenticated ro and let's also include ude the roll from the claims here within a name value pair named roll here so claims roll here like that and then instead of this is the opposite of C.abort we can include C do next so what that means is it will continue to execute the targeted endpoint which may which will be a protected endpoint excellent and that's our middleware created brilliant okay so the next step is to tidy up our routter related code so we're going to sort out the code here we want to create code for the facilitation of separating our unprotected roots from the protected roots the difference of course between these types of roots is that protected roots require the user to be authenticated before accessing a protected route and an unprotected route does not require authentication before the relevant user can access the unprotected route so let's say we want to protect the movie endpoint here the and the ad movie endpoint and the register and the login and the movies endpoints are unprotected roots the movies actually are displayed as soon as our website is accessed so we want those to be publicly available um and of course in order for a user to register this has to be publicly accessible and of course to log to the website to actually authenticate the user needs to have access to the login functionality so these this one and these two and this endpoint must be unprotected while these should be protected so we only want users with special privileges to be able to access these endpoints here so to separate our two types of roots protected and unprotected let's create two separate files within the roots folder so within the roots folder let's create a file called protected roots like this.go okay and that will house the functionality to protect our protected roots and let's create a file called unprotected roots unprotected underscore roots like this.go okay so unprotected roots and let's go to the protected roots file and the first line of code will be this package roots because we want this functionality to be part of a package called roots and we'll do the same for the unprotected roots they must be part of the same package called root so let's go back to protected roots let's create an import section here okay so firstly we want to include our controllers package within the import section so so let's go to the main.go file and I think we've already got a reference here so let's just copy this code to our clipboards let's go back to the protected roots file and include the same importation code within the roots package here within the protected_roots.go file here and you can see we're aliasing this package with the term controller and because we want to protect our roots that we're going to configure in this file we also want to import our middleware package so to do that we can simply go to go domod copy this root path here copy this root path here go back to protected roots and paste it here put these within quotations and then forward slash middleware like that and we can just verify that that is our package name by going to the relevant file yep middleware okay go back here make sure we've got middleware we are importing middleware here and that looks correct and of course we also must make sure that we've got the ginonic package installed so if we go to one of our controllers one of our controller files we should have the Jin Gonic package being imported here and we do and that's the code we want there let's copy that to our clipboards go back to unprotected roots and paste it here great so now we've we're importing the appropriate packages so that we can create our protected root configuration code let's create a function called setup protected roots so to do that let's type funk setup protected roots like this and then this uh this method signature or function signature must include a parameter of type jin like this okay and then let's firstly configure the fact that we are protecting our roots and we can do that using the routter variable the routter sorry the routter argument that's being passed into this method and we use the use method like this and then we pass in middleware dot or middleware and of course our orth middleware is the functionality we wrote in the last section of this course which will protect the relevant roots because it'll abort if the token is invalid if the token is deemed as invalid or the token has for example expired it will send an unauthorized HTTP status to the client and with an appropriate error message and then it will prevent the targeted endpoint from being called through this line of code here C.abort and that's called on the Ginggonic context so that we use we're leveraging the Jinggonic web frameworks context to abort the execution of further code the execution of the targeted endpoint like that so this is how we are protecting our protected endpoints through this middleware so it intercepts the code you can see we're intercepting we're calling this before we're configuring any of our protected endpoints to ensure that the token is valid great so now the next step is just to configure the other endpoints and we've actually already written the code for those protected endpoints so let's just we can just cut that there this is one of our protected endpoints and we can paste it here and now we're configuring our movie endpoint here which is a protected endpoint and it is protected by this line of code here the execution of code will abort if the token is invalid so it won't get to this line of code here okay and the other one we want protected is the add movie endpoint so let's cut that let's go back to our protected endpoint configuration code here and paste it there brilliant so that's pretty much it that's our configuration code set up for the protected endpoints like that there and now we want to move on to our unprotected endpoint so actually what I'm going to do to make this simple is to just copy this code here and okay we can just select all here and paste it in here and the only one we don't actually need is the middleware so we can remove that importation code here because these are unprotected endpoints and this function should be called setup unprotected roots and of course these roots are protected roots so we don't want these configured here we're already configuring them here within setup protected routes so setup unprotected roots let's firstly remove those and then let's go back to our main.go method where we are currently configuring our roots let's copy the unprotected roots or cut the unprotected routes so register and login and let's place these here and we've got one more unprotected route to configure and that is the movies endpoint which is publicly available on the homepage of our web application which just displays all the movies that are currently available to be streamed on our Magic Stream website so let's go back to unprotected and let's configure that here and there we go and that's pretty much it so the next step of course we have to call these two functions from main.go so we need to import our roots package in main.go firstly so let's do that so we can just make a copy of this here and replace controllers with roots which is the name of the package that we want to import here roots so now we just need to call the routter dot setup unprotected roots method like this and pass in the routter object here which is from the Genonic web framework so this code is enabling us to manage our roots through the Genonic through leveraging the Genonic framework um so I'm calling routter here that's why I've got a red squiggly line this should be roots because it's the package name setup unprotected roots not the routter um returned from the Genonic not the routter we're leveraging from the Genonic web framework this is our roots package and this is the method we are calling here so let's go back to main.go Go roots dots setup unprotected roots and then we want to call roots do setup protected roots like that there okay so now we want to test our functionality so let's run our code so let's go go run dot to run our code and let's see if we've protected those roots effectively okay so it's listening on port 8080 let's launch Postman let's launch Postman okay and I've already got this set up here but um firstly I just want to do a test that fails so if we go to movies sorry movie and then forward slash and then we want a particular movie so let's choose the hangover which has an IMDb ID of TT111 9646 9646 like this okay forget the body here this is going to be a get we're going to make a get request so um so this body won't have any relevance here what we've got we've still got this left over from when we logged into the system when we tested the login functionality um so I'm just going to go out of that there so we're doing a test for a get request to the movie endpoint and we're passing as a parameter to the movie endpoint this IMDb ID so it should return the relevant data for the hangover the movie the hangover which is uh which has this unique IMDb ID so let's try that let's press the send button and see what happens okay and that's what I would have expected unauthorized it's because we have protected the movie endpoint so you can see here protected roots and this functionality is being called this functionality to validate a token which we haven't which we were not sending through with the last call that's why we got this unauthorized 401 status response HTTP status response from our code here because basically there was no token so it would have aborted no token provided let's have a look at the error that we got let's have a look at the error we got in Postman authorization header is required so that's right there would have been nothing in the header we haven't configured that so in order to configure the header correctly whereby we pass in a valid access token we need to firstly log into the system extract the relevant token and then include that in the authorization header and we can do that with this setting here from within Postman to test the authentication functionality through a bearer token okay so to get the bearer token we need to firstly log in to the system so let's go to the login endpoint and we need to perform a post request here let's go to our body here and we've already got the relevant credentials we can use to login already here for us in JSON format within the body raw setting here within Postman so we got Craig Denton's credentials so let's log in okay brilliant so that's brought back Craig Denton's details for us and now we have we should have a valid token here because we've authenticated with our system through Craig Denton's credentials we can now use this access token so let's copy this access token to our clipboards i'm just going to paste that into Notepad here now let's um include the relevant URL here for the movie endpoint and let's try again and so TT 111 96 46 this is the IMDb sorry 46 this is the IMDb number for the movie The Hangover so we want to we want the details for the movie The Hangover to be returned from this request and of course it's a get request okay and then we can configure our authorization header by going to the authorization tab here and within bearer token so we need to select bearer token here and then within this token field we can include our bearer token this is the token that we just pasted into notepad after logging in as Craig Denton so let's copy this token let's go back to Postman here and let's paste that token within this field here and of course make sure that this O type field is set to bear a token so make sure this dropown is set to bear a token then paste in the token that has been returned to you once you have authenticated with the system through Postman just copy that paste it into bearer token make sure that you're calling the correct endpoint here which we are here movie and then the IMDb number this is a protected endpoint so we need to include a bearer token a valid bearer token now let's see if that works so we've already proved that our orth middleware has protected our endpoint because it returned a 401 unauthorized error to us when we tried to call this endpoint because we didn't include a valid access token and we are doing that now through this mechanism here provided to us within Postman okay so now we've included the token and let's see if it gets validated and returns to us the relevant details of the movie hangover denoted by this IMDb unique identifier here so let's press the send button look at that 200 okay so our token authentication functionality is now working excellent and we've cleaned up our root code we've cleaned up our root related code here by separating protected roots which we'll continue to include within this function here from unprotected roots which we'll configure within the setup unprotected roots function here excellent okay so we've separated our endpoint roots into two main categories protected roots and unprotected roots protected endpoints can be accessed once a user has been authenticated and has the appropriate security privileges the protected endpoints we have dealt with so far are automatically accessible once a user has authenticated but for the endpoint that we are going to work on now a user must not only be authenticated but must also be a member of the admin role this endpoint will be created to enable administrators to update a movie review for a specific movie for the sake of simplicity only one movie review will be created per movie and this must be done by an administrator the exciting part of this functionality that we are about to create is that we are going to use a technology called lang chain go to interact with open AI and extract the sentiment of the movie review which of course will be written using natural language so basically we'll prompt the AI to use one of five words excellent good okay bad or terrible to describe the sentiment behind a movie review written in natural language i've deliberately kept this example simple so that the basic principle of leveraging AI to extract a quantifiable metric from natural language can be easily understood so to elaborate on the significance of the use of AI here basically each of the words I have chosen to describe the sentiment behind a movie review has an associated number excellent has the associated numeric value of one good has the associated numeric value of two okay has the associated numeric value of three bad has the associated numeric value of four and lastly terrible has the associated numeric value of five if we look at the movie documents in the movies collection of our magic stream MongoDB database we can see we have the admin review field which contains an example of an administrator's movie review which is of course written in natural language then we have this ranking JSON object here that has two fields namely ranking value and ranking name ranking name will contain the word that through the use of AI will be extracted from the administrator's movie review to describe the sentiment behind the relevant movie review ranking value will contain the numeric value associated with the word extracted from the relevant movie review to describe the sentiment behind the review so a simplistic example would be the administrator's review says something like I loved this movie the acting was simply sublime in this particular case we've got I did not like this movie so an even more simple example and the ranking value is four and the sentiment is bad which is denoted by the ranking name so the AI has extracted the word bad to describe the sentiment behind the movie review so bad is included as the value for the ranking name and the value of four associated with bad is included as the ranking value so the ranking value essentially can be used as a quantifiable metric pertaining to the sentiment behind the review written in natural language if however the review went something like this I did not love this movie but I did not hate this movie either the associated ranking name would probably be okay and the ranking value would therefore be three which is associated with the sentiment denoted by the word okay so here we have a similar admin review to the one I've just described this movie wasn't great but it wasn't bad either and of course the ranking value for that is three associated with the sentiment which is denoted by the ranking name of okay and we're using AI to extract these values from a review written in natural language and this is how we can quantify a review written in natural language and you'll see later how this pertains to the recommendation service that we provide through the magic stream website okay great so let's write the relevant endpoint HTTP handler function code right so let's go into our code and start writing the logic for the relevant code which is to do with updating a review for a particular movie so we're updating the admin review for the movie then a call is made to open AAI with which will contain a prompt and the review and then the functionality that we'll leverage through lang chain go will extract the ranking value and the ranking name from AI which basically describes the sentiment behind a particular movie review let's go to the movie controller here let's go to the bottom here okay I'm going to call this function admin review update like this let's hook into Jgonic as we've done many times before now handler funk like this oops let's create our anonymous function that we'll return from this function okay let's include the context as a parameter here like that and let's write our logic so firstly let's create a variable named movie ID let's make this camel case movie ID let's use our assignment operator here which also declares the type of the movie ID variable based on the inferred type from the parameter passed through to our function handler here or our handler function so it's going to pass in a parameter named imbore id so movie ID this will be the IMDb unique identifier for the movie that will be passed in with the URL to our handler function here okay and then we firstly need to check whether our movie ID indeed contains a value so if movie id is equal to an empty string then let's handle this specific case so we want to send back a particular HTTP response to the user so HTTP status bad request so this would be a bad request by the client if the IMDb ID parameter value is empty is not included within the URL okay and let's send back wellformed JSON so jin.h open the curly braces there include error here which is the name and the value of the error is movie ID required so it hasn't been included for whatever reason here and we are returning an appropriate message via HTTP okay so we need to include this code of course here um within the return function okay so and then we can include the return keyword here so that the code halts at this point if no IMDb parameter value was included in the URL we don't want to continue with the code okay let's create a local strruct named req we'll go var like this strruct and let's define the strct and it just contains the admin review an admin review field of type string and let's create the JSON tag here because we want this field to look like this within admin code within sorry within JSON code review like that okay and the red squiggly line just means we're not currently using it okay and then we let's create another strct another local strct and let's call this one rest short for response strruct like this and then ranking name is our first field string and we want ranking name to look like this within the JSON so ranking name like that and then admin review we're also going to be returning to the user and you'll see the significance of this when we write the client code client react code JSON and then admin review like that perfect okay and now we're going to bind the passed in body this is going to be related to a patch HTTP request from the client so the body will contain the field and fields and values that we want to update within the movie within a movie document within a targeted movie document denoted by the unique IMDb_ID parameter value so let's type if declaration C let's use the context and then we've used this before should bind and then let's include the amperand which relates to a specific part of memory where the data for this strct is stored so amperand w colon error we need to check the error now in a if condition error is null sorry is not null then we want to handle the exception so we go c.json JSON and send back an appropriate HTTP response which is the same as the previous error handling function uh so it's a bad request here and jin.h error invalid request body so we were unable to bind it to our wreck strct here whatever was included in the body so we must send back a bad request response from our server code with an appropriate message invalid request body there and let's as usual halt execution with the return keyword like that okay great now we're getting to the exciting bit where we're going to use AI to extract the sentiment of the admin review that will be passed in to ultimately bound to our strct here which is the admin review be passed in through the HTTP body through the HTTP message within the body of that message and we're going to bind it to the wreck which we're doing here and if it gets to this code the binding has been successful okay and now we want to pass that admin review sorry this admin review here into open AI using lang chain go we're going to use lang chain go for this and I'm going to show you how we can do that now and then it will return based on our prompt it will return the sentiment okay okay so firstly let's sort out our prompting code okay so let's go to env we're going to store a prompt a base prompt because the the ultimate prompt is going to include our base prompt and the administrator's review so let's first include our base prompt here this will this base prompt will be included with every prompt so it's a generic part of our prompt and then the other part of the prompt will be the actual admin review so so let's create the base prompt within our env file here okay so let's call this base prompt template like this equals and then the prompt is return a response using one of these words and then within curly braces we include a placeholder called rankings and this will be added dynamically we'll first query the database for all the rankings that are available so you can see here not ranked as a out of limits value 999 not ranked and then we've got one excellent two good three okay four bad and five terrible so this makes it extensible so we can add to that ranking list if we want to within our database and then that will be added to our prompt dynamically through our code okay great let's finish our prompt and then we want to say the response should be a single word and oops and should not contain any other text so we're being very specific in our prompt here the response should be based on the following review and then colon and then we'll include the review we'll concatenate the review to the string uh denoted by the base prompt here we'll concatenate the administrator's review to the base prompt and send that through lang chain go to open AAI and that will return a response a single word which will either be excellent good okay bad terrible and you can extend the values included within the prompt by adding it to this collection here the rankings collection here at the moment we've got one excellent two good three okay four bad and five terrible okay excellent great so let's go back to our code here so let's create our function that will encapsulate the functionality for prompting the AI so let's create a function down here funk let's call this get review ranking like this and then let's include the admin review as a parameter here a string and we want to return a string which will be the sentiment denoted by one word excellent good okay bad terrible int will be the value associated with the relevant word that denotes the sentiment and then of course an error object like this if one occurs that will be relevant to the way the client handles the return from this function and now we need to create a new function which will get all the rankings so it's going to query our rankings collection and get all of these rankings and so that we can add them to our prompt our base prompt within the placeholder here denoting the rankings so we want to get a list of those and we're going to add them as a comma delimited string within our prompt here okay okay so let's go back to our code yeah so we need to create a function now here called get rankings let's go funk get rankings like this okay so this get rankings function will return an array of models.ranking ranking to the calling code and an error object so let's define a local variable called rankings and this will contain a collection of all the rankings that are saved to our rankings collection within the MongoDB database so models dotranking like that okay let's do the usual let's create the context the resource handling functionality so ctx cancel equals to context dot with timeout and then context dot background okay and then let's include the defer cancel code here like this and now let's create a cursor variable because we want to return the collection from our rankings collection within the MongoDB database so we want to return all of these documents so we want to include all of the items within this rankings collection all of the documents from a particular query so let's go ranking collection and we haven't created the ranking collection yet so to do that let's go up here and let's create the ranking connection variable and similar to this code but not exactly the same so we call this ranking collection here and we're querying the rankings collection within our MongoDB database let's take that let's copy that to our clipboards here so ranking collection so that we can query the ranking the rankings collection within our MongoDB database dot find like this let's pass in the context sorry pass in the context and bon m which means we want all the documents returned and put into our cursor here all the documents returned from the rankings collection and put into our cursor variable there like that let's check for any errors like this if error not equal to null then we go return null and like that and then we can write this functionality to make sure that the cursor is closed so this is also housekeeping functionality to make sure that resources are cleared up pertaining to the cursor variable so cursor dot close and ctx like this okay and then if assignment operator cursor do all cct so we're putting in the cursor whatever is extracted from the cursor we are now putting that what whatever is in our cursor we are now putting that into our rankings variable at a particular memory address okay okay so that's the rankings variable we defined up here as the strruct ranking okay and then let's check if is not equal to null let's handle that error and we just return null because we can't return anything if an error occurred and let's return the error object to the client and then if it gets to this point in the code that means all has been successful and we have successfully extracted our collection of documents ranking documents and put them into our rankings array here so we want to return that to the client and we can do that with this code here rankings nil like that okay brilliant and then let's go back to our calling code which will be the get review ranking function so rankings whoops rankings uh operator equals to get rankings like that okay and then let's check the error object and if it does have an error we want to return nothing to the calling code zero and error like that okay however if an error hasn't occurred we're going to create a variable called sentiment delimited like this colon equal let's just initialize it to an empty string and then let's create a for loop so for underscore ranking operator equals to range so we want a range of values rankings the range of values stored within our rankings variable open that there if ranking dotranking value is not equal to nine whoops 999 then we want to include that within our sentiment delimited which is a comma delimited string we want to include the ranking name within our sentiment delimited variable so you can do sentiment delimited equals to sentiment delimited plus ranking ranking name like this plus and then a comma like that great okay that's looking good okay and then we want to trim off the last comma and we can do that with this line of code so we can just go oops sentiment delimited equals to strings like this strings.trim and I don't think we've yet included strings within our importation code so we must do that yeah so let's include strings here okay go back here to our code strings.trim and then sentiment delimited and we want to trim off the last comma like that excellent okay so now what I want to do is load the base prompt into memory so we're going to read this value here into memory into a variable in memory right so let's do that let's go equals to go dot env.load like this we got to load our environment file here environment variable file so we need to import the package with this functionality to read the env file so let's go up to our importation code here and include this we need to include this particular package here like that go.v here okay and we're loading the environment file if it exists and then we can just do a quick check to see if the file exists this actually only applies to when we're running our code locally we're using thev file but when we deploy this we'll use the hosts facilities to set up the environment variables in the cloud so this only applies this code here only applies when we're running it locally for test and development purposes so this error will actually occur when the code is running in the cloud so all we want to do is just log the fact we don't need to stop the code or anything like that because it's not a mission critical error so it's just a warning we'll give it's just a warning that we'll log here env file not found so in some cases this will be an expected error and we need to import the log package here and we can do that with this code like that okay excellent we now want to read a particular environment variable which is actually going to contain our API key we we have to have a valid API key in order to communicate with open AI so we're going to be sending our prompt to OpenAI in order to extract the relevant word the sentiment associated with particular movie reviews so if you haven't done so you must sign up for a an OpenAI platform account so to do that we go to openai.com like this and then we go to the login dropdown here and then we go API platform so you must log you must sign up for an API platform account and to do that we click this particular option here okay and if you haven't yet signed up just go to the sign up here the only downside to this is you're going to have to pay a minimum of $5 but the good news is that $5 lasts a very very long time so with each request you get debited a tiny really tiny amount so your $5 will last you a very long time so that's the base amount the minimum amount you have to pay to uh to be able to prompt the AI in the way that we want to in this application so you will need to sign up for an account and pay a minimum amount of $5 in order to use the the facilities provided by OpenAI platform here so I've actually already signed up for an account so I'm going to log in with Google with my Google credentials okay and then what you want to do is once you've signed up you'll have a dashboard so you go to the dashboard option here and then you want to create a new API key and this is what we're going to configure within ourv file an API key that we can use to communicate with open AI uh and we're going to do that via a technology called lang chain go so you can see I've created a few keys here uh just to for demonstration purposes I'll create another one so you'd go create new secret key and let's I'm just going to name this um uh I'm just going to name this magic stream key so you can name it whatever you want and then you just go create secret key like this and then you copy that secret key to your clipboard like that and then let's go back to our code here and we can configure that within thev file here i'm just going to call this setting open AI API key equals and then I'm going to paste that key in here like that and uh so obviously I'm going to deactivate this key once the course is being finished so you need to create your own a your own new API key through the open AI website so that you can follow along with the next part of the course where we're going to prompt the AI using a custom prompt in order to extract the sentiment from a movie review so I urge you to create your own API key and then configure it here let's go back to our code okay so firstly we need to install the lang chain go package so to do that we use the go get command and then followed by the path to the relevant package which is github.com/tmc/lang chain go uh slm/op ai like that press the enter key so we're installing the lang chain go package and that is installed successfully and then we can just include the relevant package importation code which is this code here then we are importing lang chain go here like this so this is the code for importing blank chain go into our application and we're about to use that package now to prompt open AAI okay so let's continue okay okay so we're now going to communicate with OpenAI and we need our unique key which we have just saved within the ENV file here okay let's go back here okay okay and let's write the code to read that key so open AI API key operator assigned operator OS dot and we're reading the environment we're reading the particular environment variable which we've named open AI API_key like that okay it's saying that OS is undefined so we need to import OS also okay great okay so that will read the environment variable into our open AI API key variable there let's just handle the case where no environment variable is available for this particular environment variable name if no value is available we need to handle that so if open AI API key is equal to empty string let's return the following to the calling client code zero errors dot new could not read and then the name of our environment variable able here like that errors and we don't have errors imported let's go up here and import errors and we'll import it here errors and let's now because we we should have our open API key now read in from our environment variable then we should be able to communicate with the AI using the lang chain go package so we go open AI new like this and then open AI and then oh sorry AI with token like this and then we can include our open AI key here in order to communicate with open AI and you leverage their functionality so we've signed up for a valid Open AI key and we are now passing that through lang chain go to open AAI so that we can leverage their services here so that we can extract a sentiment from a movie review which is denoted which is going to be included within a custom prompt that we'll use to prompt open AAI okay and then we check if error is not equal to null return empty string zero error so the ranking name we haven't found a ranking name because an error has occurred zero is the ranking value and then the actual error will be returned to the calling client now we want to read our base prompt template so we want to read we want to read in this value here base prompt template like that we want to read so we want to read in this value here the value here for the base prompt template environment variable so let's go back to our code here let's create a variable called base prompt template and then assignment operator OS dot get env okay and then let's include our key which is base prompt template like that okay and now we want to now that we've read in a delimited string containing all of the available ranking names or the sentiments available excellent good okay bad terrible and a delimited string we want to replace let's go to env here we want to replace this with the actual delimited list of sentiments or ranking names So let's go back to our code here and let's do that so we're extracting the base prompt here and now we're going to replace that value with the value extracted here that we got from the get rankings function that we created okay so let's do that let's go base prompt so we got a new variable base prompt this is the template and we're going to replace in curly braces rankings with the delimited list of actual ranking names it should be okay and let's go strings dot and let's use the replace function and the first argument to the replace function is the base prompt template variable followed by the value that we want to replace within our base prompt so rankings like this and then the delimited string sentiment delimited one like that and that is how we replace this placeholder value within our base prompt with the actual list of delimited ranking names or the various sentiments excellent good okay bad and terrible okay now we're going to actually call OpenAI we're going to leverage their API functionality through Langchain Go so we go llm.all context background oops that's not what I meant to do context dot background that's more like it and then we pass in our base prompt but we also we don't just want the base prompt because if you look here at the prompt itself the last thing we're saying is the response should be based on the following review so we want to concatenate the actual review an actual movie review okay so let's go back here and let's do that and we can concatenate a string by using the plus operator and then the value that we want to concatenate is the one passed in as a parameter value to this get review ranking function so we do this whoops where am I plus review like that okay and then we want to do the usual thing and check if an error was sent back if it's null if it's not equal to null let's handle the error and we just want to return an empty string zero for the ranking value so the empty string represents that there's no ranking value extracted here zero no uh no ranking name extracted here zero representing no ranking value and the actual error object gets passed back to the calling client like that and let's initialize rank val here to zero okay and we need we're going to do a verification here with a for loop to check that the ranking sent back from the prompt that we've sent to Open AI is indeed in our rankings collection so we're going to loop through the range of rankings like this so all the rankings excellent good okay bad ex uh bad terrible and we want to test through an if statement whether that ranking exists whether the ranking name sent back the sentiment sent back from open AAI is valid response like this so the response will contain the sentiment name or the ranking name excellent good okay bad terrible and we need to loop through our collection to make sure that what is sent back is valid and we go ranking and then we can also extract the ranking val so we're quantifying this is how we're quantifying qualitative data which is the movie review so the qualitative data gets sent with the prompt so ranking val equals to ranking dotranking value like that and then we want to break out when it's found when the response is found we want to actually break out of this for loop here so we can use the break keyword for that and then we return response rank val and nil for exception because if it gets to this point everything has gone well and we can now return the ranking value sorry the ranking name excellent good okay bad terrible and its associated value which is a value from 1 to five one being excellent terrible being five and we're returning that to the client there and that's our code for the get review ranking so here we've interfaced with open AAI using lang chain to return a quantifiable value from qualitative data in the form of a movie review and we're returning the ranking name and the ranking value to the calling client code so let's go back up to the actual admin review update function and let's call our get review ranking function okay great so let's call it sentiment so this is the value we want returned the actual sentiment and this is the value the ranking value and the error object assignment operator and then let's call our function get review ranking like that and then wreck so this is passed in to our HTTP patch request the admin review is passed in through the body of the HTTP message we've extracted it and we've bound it to the recstruct here and we are using the admin review field that is then passed to the get review ranking function so this is part of the prompt the admin review is part of the actual prompt that gets sent to OpenAI and it returns the sentiment associated with that admin review excellent and also of course the ranking value quantifiable value from 1 to five then let's check for an error not equal to null like this and then if an error has occurred let's pass back an internal server error or 500 HTTP response to the client status internal server error and then let's pass back wellformed JSON for the error message like this and the error is error and the actual message is error whoops error getting review ranking like that okay excellent and then let's include the return keyword here so that execution halts if an error occurs at this point here okay okay so now we're actually going to include the update code that updates a movie document within the MongoDB database and that movie document is based on the IMDb parameter value that will be included in the URL to target this functionality here add admin review update functionality okay so let's create a filter we want to filter on the IMDb parameter name so bson M open curly braces let's include the parameter name IMDb ID and then our movie ID variable which will be the actual IMDb ID unique value that denotes a particular movie on the internet and let's go update so this is the actual update information that we're going to be relaying to MongoDB telling MongoDB which fields we want updated within our database with which values okay and to do that we use the set command like this so it's dollar set i've used percentage not dollar that's not good we want dollar here okay and then colon bon m oops m open curly braces admin review colon rec and then the actual admin review so we want to update the admin review and then we also want to update the ranking object within the movie document so we are updating the ranking object with this code so we go bon oops do m and we open the curly braces and let's update the relevant fields so ranking value let's include the value which is rank val here and then let's include ranking name colon and this will be the sentiment extracted from openai we actually need to include a comma here and then a comma there and a comma there like that and that's the update code and now we can actually include the code to interface with our database so let's include the usual i'm just going to paste this in here because we've written this so many times now and this is just for cleaning up the resources after we've performed our update functionality against the MongoDB database okay so we're going to use the update one method on the movie collection variable for this purpose for the update to be executed result assignment operator movie collection dot and we go update one because we're only updating one document and we include the context cleaning up the resources filter to target a particular document within the movie movies collection and then the actual update information here we've stored in this variable the update that needs to be the update information that needs to be passed to MongoDB to describe which fields need to be updated and with which values okay great and let's just check if any errors have occurred for the update so if error is not equal to null let's pass back an appropriate error response to the client so HTTP response of status internal server error 500 will be sent to the client in this particular case if the update has failed h so we go error and then colon whoops colon error updating movie like this and then let's include the return keyword there okay okay and we also need to check the result returned from this operation here this update operation so if result dot match count so it hasn't found it hasn't been able to target a particular document based on the IMDb value passed in as a parameter value to this handler function method it hasn't been able to find that document so if that equals to zero there's no match sent within the result of this operation we need to handle that case so C do.json and then the HTTP response is that the resource was not found so HTTP the resource we wanted to update was not found so we pass back to the calling client status not found like this and then let's include an appropriate error message and let's use jin gonic to pass back a well-formed well-formed JSON data error colon movie not found brilliant and then of course we need to include return keyword so that our code halts because it hasn't been successful if the code gets here our function handler our handler function method has not been successful if the code gets to this point the document wasn't found so we haven't been able to update the sentiment and the ranking value the ranking name and the ranking value fields or the admin review within our database however if the code gets to this point everything has been successful let's create a response so risp dot ranking name so we're setting the ranking name to the sentiment variable and restpadmin review equals to rec and then we can send back an appropriate response c.json we want to send back a HTTP status okay response if the code gets here all has been successful and then let's pass back whatever is in the response strct which should be the ranking name which would be excellent good okay bad or terrible depending on the sentiment extracted from the movie review extracted by the open AI functionality that we've been able to interact with through the lang chain go technology great and that is our code written so in this part of the course we're going to create a movie recommendation service for our Magic Stream web application we have just created a facility to update reviews for movies and we've also implemented the AI functionality that extracts the sentiment denoted by one word from the movie review so in our recommendation service we can use the ranking value to recommend five movies and order how the movies are displayed to the user by those ranking values so we can order the way the movies are displayed to the user in ascending order from one which is mapped to the sentiment denoted by the word excellent to five denoted by the word terrible so how the recommendation service will basically work is our code will query the movies collection based on the user's favorite genres that the user chose when registering with the magic stream application then we can select the top five movies from those genres which will be ordered by the ranking value associated with each of the movies so our recommendation service recommends the top five movies in the users's favorite genres and displays the movies in a ranked order from one to five so yes it's a very basic recommendation service that we are integrating into our Magic Stream application i thought this would be a great opportunity to integrate AI functionality that extracts sentiment from a review written in natural language and then quantifies that sentiment and we can then use that sentiment value to rank the relevant movies and in doing that recommend the top movies that pertain to the relevant users favorite movie genres as described earlier a review is added to the system by an administrator and AI chooses one word from a list of options provided in our prompt to the AI the word to describe the sentiment at the moment can be excellent good okay bad or terrible each of these ranking names or sentiments has an associated value stored in the ranking value field excellent the value for the ranking name has a ranking value of one good has a ranking value of two okay has a ranking value of three bad has a ranking value of four and terrible has a ranking value of five and you can of course extend the sentiment or ranking options if you like but for the sake of simplicity let's stick with the aforementioned five rankings okay so let's go to the movie_controller.go file and let's create an endpoint handler function let's first go to the bottom here and let's create an endpoint gingonic handler function and let's name this get recommended movies like this handler whoops handler funk of course oops and of course we need to return a function from here funk that accepts C star the genuine context as an argument dot context like that open up curly braces and now we can write the logic for our get recommended movies or movies as I put here forgot the V movies okay a user has to be logged in in order to use this recommendation service so let's first extract the user's ID from the Genggonic context remember in our middleware that protects our endpoints if the user security token which is a JWT token is validated we are setting the named value pair value within the context object provided to us by the Genonic web framework to store the user's user ID value so we can now use this value to query the user's collection for the user's favorite genres which the user chose when registering with the application so if we look at our orth middleware here you can see that we're getting the access token there and then we're validating the token here and then if the token has been validated we're setting these two named value pairs like this from the claims so we're setting the user ID which means we can extract the user ID and subsequent code and use that user ID in this case we're using that user ID to query the database for the genres that the user chose when registering with the system um and we can do the same for the role so we can check what role the user is in also okay great so let's go back to user_controller.go here and let's write the logic sorry movie_controller and let's write the logic for our get recommended movies function so to extract the user id value from the context let's create a function named get user ID from context within the token util file so within the utils package so before we write the logic here let's actually write a reusable function within the token util.go Go file here and let's call this function get user ID from cont whoops context like this okay oops so this function is fairly straightforward so we need to pass in we need to include a parameter of star jin.ontext context here so that we can use the context to extract the user ID from the named value pair that we saved once the user is authenticated when the user calls a protected endpoint and is authenticated we're saving the user's ID from the claim stored in the token we're saving the user ID to the Gengonic context so now we can extract that uh value from the Genonic context so let's do that user ID comma exists so the exists this is clearly going to be a boolean value this exists value so it's returning two values from C.get from the C.get method and then user ID which is the name of the value that we want to extract from Jinonx context here and then let's check the exist boolean value to see if the user ID value exists within the context okay so oops sorry no we don't need that and then let's open our brackets so we're saying if not exists so we're checking a boolean value which can either be true or false so if it doesn't exist return an empty string and then let's return an error and the error we can return can be something like this user ID does not exist in this context okay us and the reason why we're getting a red squiggly here is because I haven't included in the definition what we are returning from this function so we want to return a string which will be the user ID value and we want to return an error object so that the client can handle any errors that may occur during this code okay great then let's proceed id okay so this what we're going to do here is return the user ID as a string so we can do that with this code and if all has gone okay so we need to check the okay boolean value here if not okay we want to handle the exception and this code will be similar to the one we've already already written we're just handling potential errors that may occur here and we'll just go unable to retrieve user ID unable to retrieve user ID here so if it's not okay if this code fails we want to return an exception to the calling client else if it gets to this point in the code we can return the user ID and because no errors would have occurred we can return null for the error object so we're returning those two values to the calling code string error string will be the user ID null or error in this particular case there okay let's save that and we can now go back to the logic here within the get recommended movies and we can extract the user ID from the gingonic context and we can do that by calling the get user ID from context function okay so let's do that let's go user id and then assignment variable utils we're going to use the utils package which we have imported already into this package get user ID from context and we're passing in the gingonic context here so that we can extract the user ID value from the gingonic context okay and then we want to do the usual go thing and check for any errors so if error not equal to null then let's handle that so an error has occurred c JSON HTTP status bad request and we've done this a few times at this point and then let's pass back an appropriate error using jin.h so this will pass back wellformed JSON to the client and the client can handle the exception id not found in context excellent okay so now we want to query the database for the user's favorite genres now that we've got the user ID we can query using that user ID to target a a specific user document within the users collection and we're actually going to create a method called get users favorite genres we're going to create a function called get users favorite genres to extract the users's favorite genres before we proceed with the get recommended movies logic let's create the new function below get recommended movies okay let's create a function called get users favorit genres like this which accepts one argument of type string named user ID and returns an array of strings which will contain the user's favorite genres and an error so the client can handle any errors that may occur okay excellent and let's write the logic for this so let's first create and I'm not going to explain this at this point because we've done it so many times in this course we're just creating a context which helps clean up resources and then we're going to create a filter for our query assignment operator bon we're creating a filter and we want to filter our query on the user's ID so user ID like this and then we're passing in the user ID here as an argument and we want to include that within our filter here okay and then we want to create a projection variable like this bon m okay okay and the projection must include favorite favorite underscore genres whoops dot genre name so favorite genres do genre name so we want to include that one there and we don't want to include the underscore ID within the projection okay and we can set this to zero like that and these red squiggly lines just mean that we've created we've declared the variables but we're not currently using them but we're about to use them anyway so okay and then let's set the options so opts assignment operator options dot find one and then set projection so we're setting the projection from the variable we've created here check okay and then so options this might not yet be imported into our code we need to import a package there for options so let's go up to our importation code and import the relevant package so that we can use the options functionality there and it's just a case of including this package here which is part of the MongoDB forgo driver functionality so we're including that there and we can now use the options functionality down here within our query okay and you can see the red squiggly line under options has now disappeared okay brilliant let's continue with the logic of our code okay and then we need to call user collection.find one so we're targeting the user collection oops we're targeting the user collection and we want to find the relevant data based on the criteria we've established through these variables the options the filter the projection so let's pass those in to the find one function so let's pass in the filter and let's pass an opt here and then we decode those results into a variable that we haven't yet defined called result so let's first define the variable called result so var result bon m so the this result variable is defined as bon.mm and we're the any results that are found within our query here will be decoded into this variable here so we're going decode and then oops and then we want to include a pointer so this is in fact just a pointer denoted by the amperand character preceding the variable so it means it's the data will go into a specific location in memory and the result will point to that memory address that's what the amperand denotes here okay brilliant and then let's check for any exceptions that may have occurred so if is not equal to null let's and then we want to actually nest an if statement here and check if a spec for a specific error if mongo no documents so if no documents were retrieved let's handle that will return an empty array and null so it's not an exception but the user's favorite genres were not able to be retrieved because they don't exist within the database so that's what we're doing here okay and now we want to if it gets to this point favorite genres array we want to put the result in the fav genres array variable so let's use the result to extract the results we want so favorite genres like this bum a like that and that's how we are able to define the type of the variable as well as return the favorite genres that the user chose when the user first registered with the system okay and then we can check if not okay in other words if this code is not returning the expected result that's this boolean value is set to false so we need to handle that by returning errors new unable to retrieve favorite genres for user so if we were unable to extract the users's favorite genres from the database we are handling that case here by returning an error stating unable to retrieve favorite genres for the user however if our genres were retrieved we want to return an array of genre names to the calling code so an array of strings which is going to be just the names of the genre is what we want to return from get users favorite genres and these are the genres that the user chose when signing up to our Magic Stream website so this code is pretty involved so just bear with me as I write this code it's just basically looping through the results and putting the genre name because the uh the genre object contains a value for the genre or an ID sorry for the genre as well as the genre name we're looping through all the genre objects if we go to the database here you can see the genres here so it's got a genre ID and a drama and we just would want drama western fantasy etc we don't want the genre ID returned so we want an array of genres that the user chose when the user signed up to use our magic stream website and you can see them here favorite genres for example Ben Madison chose comedy thriller sci-fi we so if it was Ben Madison that had logged on we we would get three objects returned to us from our query and we're now looping through those objects those genre objects favorite genre objects and we're extracting just the genre name putting them into an array and returning that array to the calling code that's basically what we're doing here so v genre names this should be genre names and it's a string array okay let's create a for loop loop to loop through the results so underscore just means no value needs to be returned there so it means that the method we're about to call returns two values but we want to omit the first value we don't need this value returned we just want the item returned okay range that's the range of values stored within favorite genres array fab genres array variable here and then if genre map okay assignment operator item so we're checking the type here bund and we're these types come from the MongoDB for go driver function functionality so it must be bon of oops must be of type bon.d and if that's okay so this is a boolean variable returned here so if it's of bon d the item is of bon.d okay will be true and we want this code to run so we do another for loop for lum uh the element operator and then range genre map so genre map was returned from this operation here and then we proceed with our logic here lm dokey we check the key must be equal to genre name so of course genre name if we go to compass here we can see the genre name will be comedy for example thriller sci-fi so we're just checking that that is the name of the field genre name we're checking that in this code here let's carry on so if all that checks out name okay operator lm value must be a string value that's all okay then we want genre names we want to append the relevant value extracted so the relevant genre extracted and put into our array genre names and name so this here is being added to the genre names array here brilliant and then and then we can return that if it gets to this code so our for loop would have been successful we've basically extracted the genre names from the array which would contain an array of objects which would also include the genre ID but we want the genre names and we're extracting that genre name and putting it into the genre names array and we want to return that array of genre names to the calling code and we just go return genre names and null for exception because if it gets to this part here if it gets to this area here all has gone well and we can return an array of genre names and no error denoted by these definitions here within the main definition of the get users favorite genres function so we're returning an array of genre names and do no exception excellent so we can go back to our calling code so we're going back to our get recommended movies function here and we can now call the get users favorite genres function to get the users's favorite genres that the user chose when signing up of course to use our Magic Streams website okay equals get users oops get users favorite genres and we want to pass in the user ID to get users favorite genres so we've got our users favorite genres now let's check if an error occurred if error not equal to null let's handle the relevant exception like this http status so we'll send back a an HTTP response of status internal server error 500 gen.h and we're going to pass back wellformed JSON denoting the error that occurred here if one occurred at all errors okay like that error not errors and of course I'll put a comma there because I don't know I'm out of excuses should be a colon okay and then let's include the return keyword here so that code halts at this point if the favorite genres were not retrieved okay and then let's include this code here go env of course if this is running in the cloud the env file may not exist so we'll be checking for an error here because we'll configure our environment variables differently using the hosts facilities but when we're running it locally we're configuring our environment variables within the env file so what we want to do here it's not a fatal error in fact it's an expected error if it's running in the cloud so we'll just log that fact we'll just call it a warning here v file not found okay so it doesn't really matter if the env file is not found okay var it will sorry it will matter if you're running it locally thatv file must of course be found if um running locally so recall let's create a variable called recommended movie limit vile because we only want to bring back five records from our query 64 so we're defining it as in N64 and we're going to assign it the value of five we'll configure this in a environment variable at a later point because we don't want magic numbers hanging about in our code it's just not good programming practice okay so in fact let's do that now i'm going to go to the MV file here and let's include and let's configure that environment variable here like that so recommended movie limit five okay great and now we can read that value within our code okay assignment operator OS.get env and we are reading this environment variable here okay that there um and the reason why that is kicking up a fuss is because there's a type mismatch here go is actually a strongly typed language and what we have here is a type mismatch because this variable is defined as N64 and this will bring back a string this get env code code here will bring back a string and we're assigning a variable that I've just defined as int 64 a string value so that's why we've got the red squiggly line there so this should actually be a new variable and I'm going to change this to str so it's a new variable and the red squiggly line has gone away there and this is because we're not currently using the variable but we're using this to to read the string from the environment variable into memory here okay so let's just check if recommended str is not equal to what we can do is convert strr con using this str con package so we need to import that str conf so we're going to convert the string red from the environment variable we're going to convert it into an integer using str conf using the str cov package here okay so equal str nv dot par int so we're parsing what we've read here into an int 64 data type okay and that looks good great okay let's create the find options variable now find options this is for our query that we're about to send to our database where we're going to find our recommended movies five recommended movies for the user based on the users's chosen genres and the ranking values that have been assigned to each of the movies through the use of AI through the use of lang chain go okay so find options dot set sort so we're going to want to sort the data by this key oops by this key ranking dot ranking value so um okay and we want it in ascending order so value one for ascending order like that and we've got an issue here p this is an object here so we need to include more curly brackets there like that and that's now correct right so the key is ranking.ranking value if we go to compass you can see if we go to the movies if we go to the movies collection we've got the ranking value and we've got the ranking name so we're sorting by ranking value here from 1 to five so for example if the this uh collection of movies included Highlander 2 and Jack Reacher this would appear first because we're we are sorting the movies array we are sorting the movies result retrieved from the movies collection by ranking value so this would appear first followed by Highlander 2 if for example these were included within the results so that's all that is saying there essentially we're we are ordering the returned results by ranking value so the returned movie results by ranking value in ascending order from one to five great and let's include the filter for our query okay and we're querying baston m on genre name and this must be in quotations genre do genre name like that then bson dom so we only want to print values back from the movies collection that include the relevant genres the user's chosen favorite genres and you can sort of start to perhaps understand this query now so if the user chose comedy thriller drama we only want movies returned from the movies collection that are of those genres and this is basically what this part of the query is establishing okay so then we want to include this usual context code i'm just going to copy it there this is the cleanup related code that we include when we use the MongoDB forgo driver functionality okay so let's create a cursor for our query assignment variable movie collection we don't want to count the documents dot find and now we pass in find options the filter and the context so firstly the context brilliant and then we can just check the object if not equal to null let's handle the error c.json http status status internal so we want to send back an internal server error response to the client in this particular case if our query fails and let's send back an appropriate error message in well-formed JSON format and this can just be oops comma again it's not good replace that with a colon error fetching recommended movies perfect let's include the return keyword here let's make sure that we clear up let's use the defer method to clear up our resources within memory so we're doing a bit of housekeeping here okay then var recommended movies let's define this an array of model dot dot movie so we're returning we want to return an array of movies this should be models sorry our package is called models that's what that red line that's that's why that red line was there so we're defining a variable called recommended movies of type models.mmov here okay so if assignment equals cursor do all so we're using the all method to place all the results from within the cursor into our recommended movies array okay and let's target the memory address of recommended movies like this so this variable this here points to a particular address in memory and that's denoted by the amperand character here okay then let's check the error so if it is not equal to null let's handle that error so I'm going to just copy this here because it's quite similar and it's also going to be a status internal server error if an error occurs there but we want to send a different error message and we'll just send whatever is returned by the dot error method we'll send that back to the client like that excellent and now we're in very good shape we can now if it if the code gets to this point we can send back the results to the client json http status okay so we're sending a status okay response and of course okay needs to be in capitals we're sending the a response of status okay to the client and then of course our recommended movies array in JSON format to the client brilliant and that's basically the recommended the recommendation functionality already written here okay and it's got a problem with this declare not used right so I'm missing one thing here and I noticed that the recommended movie limit val had a red squiggly line under it because it's declared and not used we of course need to use this we're reading it a limitation value of five and if that is not found within the environment variables we're actually setting that limitation um by default here with this line of code but we're not actually setting the limit for our query set sort we're setting the sort but we need to set a limit and so after set sort here because we want to limit the results to five movies for the recommendation for our user so to do that we can use the set limit method like this so find options dot set limit and then we want to pass in the integer value here this integer value stored within this variable here and where we're reading it from the environment variables we're actually converting it from a string to an integer with this line of code here so we want to pass that integer into the set limit method that we're using on the find options variable here so we're setting we're using the set limit method here to set a limit on the amount of results that we want returned from our query and we want five movie documents returned and no more than five movie documents so we can do that with this set limit method here and pass in the limit that we've defined within by default and within an environment variable called recommended movie limit so if we go to env you can see we've defined an environment variable named recommended movie limit and we've set it to a value of five let's go back here and we're passing that value to the set limit method and that pretty much means that our function is written and ready to be tested okay so we've written the functionality for our get recommended movies for our recommendation service our get denoted by our get recommended movies function here so we're now in a position where we want to test the functionality through Postman so firstly we of course need to map this to an appropriate route map this get recommended movies handler function Jarnic handler function to an appropriate endpoint path so to do that let's go to the protected_roots.go file so we may as well define the end point within the setup protected route section here because we're going to need the user to log on we're going to need to get a token to test this functionality because it's user specific we're getting recommended movies based on the user's ID so in other words the user in order for the user to get a valid user ID they must be logged into the system so this is why we may as well define the route for this even for testing purposes within the setup protected roots method here okay so let's define our end point so for slashremented movies like this controller dot and this one is get recommended movies so we're mapping an endpoint recommended movies this is part of the path to the endpoint recommended movies and we're mapping it to the function we just created called get recommended movies and that's within the protected section because we need this or middleware to run so that the user ID is saved within the genic context here and we are of course grabbing that user ID in our code here because it's a very important part of our query where getting the user's favorite genres because the recommended movies are based on the users's chosen genres that the user picks when the user registered with the application these favorite genres were chosen by the user when the user signed up to the magic stream service okay and this is the basis for the recommendation service that we're providing through the magic stream website and this function has been defined for handling the recommendation service functionality okay and you can see what we're doing in this function is we're getting the user's favorite genres within an array here and then we're querying on the movies collection for all of the movies that are tagged with the users's favorite genres here great so we've now defined our endpoint here and we've mapped it to the get recommended movies function this is the handler function the jinggonic handler function and we've mapped it to a an endpoint recommended movie so we can now test our code so let's run our code so let's go go run dot run our code excellent it's listening on port 8080 let's go to Postman let's launch Postman here so let's firstly type in the log login login endpoint like that so this is the base URL followed by the relevant endpoint the login endpoint we're sending a post request to this endpoint because we need to log in with the relevant credentials and I've already set up those credentials so we're going to log in as Craig Denton here and this is just a standard password to make things simple during the testing and development phase okay post and that's it that's all we need and let's log in as Craig Denton great excellent 200 okay the server side code for the login functionality has returned to us a valid token this is the token which contains the user's claims the claims for Craig Denton so Craig Denton's user ID can be extracted from this token when that token is passed through to the serverside code so that the recommendation service can be leveraged so that user ID can be extracted in order for the relevant query to go through and bring back the recommended movies for Craig Denton so let's copy this token to our clipboards and now we want to change this end point to recommended movies like this and then we want this to be a get request and then we must go to the authorization tab here and include our bearer token so I'm just going to delete whatever is in here and replace it with the token that we've just received after logging on as Craig Denton let's paste that in here into this field and now we should be able to leverage our recommended movies endpoint which should bring back the recommended movies for Craig Denton so let's try that out let's press the send button excellent and it has brought back Jack Reacher okay um which is tagged up with the these genres action and thriller okay okay and you can see this is the admin review we gave it fairly recently when testing the AI functionality and it's been ranked as one excellent that's why it is listed first within the list of movies returned from our query okay and then we've got Knives Out which is also which is a drama and a thriller it's tagged up with drama thriller and mystery and it's also got a ranking value of one ranking name excellent and then we've got one that has been also excellent so we've got three movies returned to us that are excellent for Craig Denton and this one's called Blitz the Jason Staithm movie and then we've got one that's just okay and it's a comedy and it's Airplane and then we've got another one that's okay so you can see that it is ordered the movies from one to five based on the ranking value so all the ones there's three ones three that have been ranked as excellent and then two that have been ranked okay so it's returned the movies in the appropriate order and it's returned five a limit of five movies to us and we can try again with a different user this time so let's log in as someone else okay let's go to Compass and find a different user here um so let's log on as Ben Madison so Benmadison benmadison@hotmail.com i'm just going to copy that there for the credentials and the password doesn't change okay so I'm going to the body here raw and I'm going to replace this email address with Ben Madison's credentials the password doesn't change so I've just made this easy for testing purposes this must be a post request so let's set that to post and let's send that to log us in as Ben Madison and we got a token that's been sent to us here let's copy the token now let's change this to recommended movies and this is a get request and let's replace the token authorization bearer token here let's select all of that delete what's in there and paste in our new token for Ben Madison and shouldn't have a quot the quotations should be omitted here okay there shouldn't be any quotations within the token so let's test our recommended movies end point for Ben Madison let's see what movies get returned for Ben Madison so let's send that through excellent okay it's got Jack Reacher again at the top and then it's got Knives Out okay blitz airplane and this time it says so there is a difference here so they obviously had very similar genres selected and then we've got the undiscovered country which was different from the last set of results and it is ordering these movies in the correct order from one to five okay for good measure let's log on as someone else so let's log on as Bob Jones who's an administrator okay so let's grab his email address here let's go to login here set this to post okay let's go to authorization set this to no orth and we're going to go to the body here replace the email address with Bob Jones's email address the password is the same and let's send that through to login bob Jones okay good and you can see Bob Jones's favorite genres here he just likes comedy and fantasy okay and so let's grab the token for Bob Jones here copy it to our clipboards and let's change the end point here to recommended movies like this this must be a get request let's go to authorization select bearer token and let's replace what's whatever's in here with the token for Bob Jones okay and now let's test bear in mind Bob Jones's favorite genres are comedy and fantasy and let's see what movies are returned for Bob Jones okay send okay so fantasy you liked fantasy and we've got Harry Potter and the Philosophers Stone and this movie's been deemed as okay based on the admin review this movie wasn't great but it wasn't bad either so it's okay three we got Airplane he liked comedy so that's correct airplane and it's also been ranked as okay and then the next movie is Gone Girl which is a drama and a fantasy okay so fantasy has been tagged for Gone Girl i'm not sure if that's accurate but in our data it's been tagged with fantasy so that's why Gone Girl has come back and it is ranked as okay so that is ordered correctly within the results the Hangover comedy three okay so they're all okays basically right so our recommendation facility is working correctly based on the data that's saved to our database obviously this is just test data so don't expect it to be 100% accurate in terms of the genres and the rankings but our code is working correctly this verifies that our code is working correctly our recommendation service is working as expected excellent so if you'll recall we created this end point and we temporarily added the code to configure the endpoint i.e map it to the relevant handler function the jinggonic handler function and we initially put this configuration code within the setup unprotected routes just for testing purposes so it would made it easy for us to test the functionality within this function here which provides functionality so that an administrator can add movie reviews to the system and update those movie reviews within the system um so if we go to compass here you can see those movie reviews go to the movies collection each of these has a movie review this is a lovely movie and it's been rated as excellent by the AI the sentiment detection functionality that we wrote which engages with AI through lang chain go brought back excellent as the sentiment behind this movie review so it's all this functionality we're concerned with but only an administrator should have access to this functionality so we don't want ordinary users to be able to add reviews to the system and that's what we're going to do now we're going to protect the relevant update review endpoint so the first step to do that is we're going to grab this code here let's cut the code and let's paste it into the protected end point configuration code here within the setup protected roots function here so now this is protected but we haven't yet written the code to properly protect this route because this is a special route if you like because the user must not only be logged on but that user must be part of the admin role so that's the code we're going to implement in this section of the course so let's do that firstly we want to be able to extract the role from the claims which we're doing here and we're inserting the role within a named value pair within the context the ginonic context which makes it easy for us to extract the role in later code um so we can extract whatever role once the token has been accessed from within the middleware and it's been validated we can extract the role this is what we're doing here we're extracting the role and placing it within a named value pair storage facility provided to us by the Genggonic context and then it goes see next and it'll move on to the targeted endpoint which in our particular case will be the update review endpoint and we'll be able to access that role through the jinggonic context okay so if we go to movie if we go to yeah movie_controller and we go to our add review functionality so we've got our admin review update function here the first thing we want to do is access what role the logged on user is part of so to do that we can let's first go to the token_util.go file and let's just create a copy of this and then let's just change the bits appropriate to what we're trying to do here which is instead of access the user ID we want to access the user's role so get RO from context is the name of this function and of course we need to change this to RO and let's make this variable RO instead of user ID so roll does not exist in this context if the RO is not if you're not able to extract the RO from the Genonic context we need to handle that error there and then let's use our RO variable here let's name this member roll camel case like that and then we can return the member roll to the calling code and that is our reusable function written for extracting the role from the jinggoni context and let's go back to the movie_controller.go file okay okay and now we can use the the function that we've just created within the utils package to extract or to return the users ro to this calling code here so utils dot get roll from context and then we must pass in the gingonic context variable here or the gingonic context argument here and then let's handle the error if an error occurs when trying to return the role the user's role let's handle that exception okay so C.JSON let's send back a HTTP bad request in this particular case status bad request oops request and an appropriate error message and we can use the gin.h functionality here to return wellformed JSON let's include a colon there and then roll not found in context okay and then let's of course include the return keyword so it halts the execution of this function we don't want this function to continue if the role is not found because the user that's trying to add a review must be part of the admin role so it's crucial to this functionality that the user's role is accessible okay and then now we can just do a check here so if roll and I'm just going to leave the code like this for now just for simplicity we could put this in a configuration file but I'm just going to leave it like that as a literal string admin and just for simplicity and let's handle the exception so if the user is not part of the admin role we need to pretty much kick the user out so that they do not have access to this functionality and it's a status bad request again okay and let's use jin.h H to send an appropriate error back to the client code and we just say user user must be part of the admin role and it's that simple and then we use the return keyword to stop the execution of the code within this function okay brilliant and that is pretty much it so now the user has to be part of the admin role in order to be able to leverage the functionality within this handler function the function that handles the endpoint configured here the endpoint update review okay so we can test that code now so now let's launch Postman and let's go to login we need to log in the user okay and we want to log in this user as Bob Jones okay let's go body raw okay great we've already got Bob Jones's credentials here and this is a post request okay brilliant okay so let's run our code okay brilliant it's listening on port 8080 and let's go back to Postman and we can test our login functionality let's press the send button here excellent bob Jones is logged in bob Jones is an administrator you can see here his role is set to admin so he's an administrator and we just want to copy Bob Jones's token here we want to copy Bob Jones's token to our clipboard let's do that copy and then we want to test the update review endpoint so we go update review like this and we want to update the review for a particular movie so let's go to the compass and let's find a movie that we can update okay and as I've said I hate this movie for um Unforgiven and that couldn't be further from the truth i love this movie so actually let's test a review for Unforgiven adding a review for Unforgiven great movie gene Hackman Clint Eastwood excellent movie so it's currently set to five terrible i hate this movie and let's create a review that is a little bit more glowing well a lot more glowing than that okay so so I'm going to copy the IMDb for unforgiven here okay and I'm going to paste it here for the parameter so that we can target a particular movie for the review and then let's set this to patch because we're only partially updating the movie document we're not replacing an entire movie document we're just replacing this part the admin review and then our AI functionality will automatically extract the sentiment and save it to this movie document this part of the movie document here okay so great we're set up to test this so it's a patch request we've copied the token to our clipboards so that's the next thing let's go to authorization here bearer token let's replace whatever is in here that's left over from the last time we tested an endpoint through Postman and let's paste the new and let's paste the new So let's paste the Let's paste the token from that Bob Jones received once he logged on and paste it here okay brilliant and now we need to update the body so we need to include a field within the JSON here admin review and let's create a glowing review for unforgiven Clint Eastwood as always was magnificent what an amazing amazing cast and story okay Clint Eastwood were Clint Clint Eastwood as always was magnificent what a what an amazing cast and story okay so that's our very simple movie review for Unforgiven and let's just make sure that this is appropriate JSON data that we're passing through so I'm just going to go back to the code here and if we go to the handler function here let's just check so yeah it's expecting admin review and the JSON must look like this admin review you can see that by the JSON tag here and then whatever we pass in here gets mapped to this wreck strct here okay within our code so that looks good and then we can just test that so we've got the endpoint URL which includes the movie parameter the IMDb value update review let's just make sure that the endpoint is 100% accurate so if we go to protected roots update review and then followed by the parameter so let's go back to Postman and that looks good we've included our review within the body of the HTTP message we're about to send through to our endpoint we've configured the token because we just logged in as Bob Jones who's definitely part of the admin group so this code should actually work and let's see if it does we'll press the send button 200 okay excellent yep i would have expected excellent to be returned as the sentiment behind the admin review that we've just created for Unforgiven brilliant so that has worked we've been granted access to the endpoint because Bob Jones is a part of the um admin role so that has worked so if we go to compass we can see now if we refresh the data reload data and we go and find unforgiven after we've refreshed the data there it is and let's look at the rankings so it's updated our review clint Eastwood as always was magnificent what an amazing cast and story and then we've got the ranking section here and the ranking name is set to excellent so the AI deemed this movie review as having an excellent sentiment and of course the value for excellent is one so that has worked perfectly brilliant so I guess the thing we can do now is just deliberately perform a test that fails so to do that we can type in login here and we can login as just an ordinary user so we've got Craig Denton so let's change this back to email and this toradent hotmail.com and the password for all of our users is just a very simple password not very secure password but this is just for testing um and it is I believe at one exclamation point like that so we're logging in as Craig Denton we need to set this to post it's a post request for the login functionality and then we can just send that over brilliant 200 okay we've logged in as Craig Denton so let's copy Craig's token to our clipboards here oops here it is okay copying Craig Denton's token here let's go to authorization and replace the token left over from the last test let's remove that and add in our new token for Craig Denton's login okay and then let's update the endpoint path so it's update re view and unforgiven let's just copy that token again we'll stick with the same movie unforgiven let's include it as part of the URL so it's a parameter within the URL that targets a particular document within the movies collection so we're targeting unforgiven with this URL here so we're updating the review for unforgiven at least we're going to try so we're actually hoping that this will fail because we want users who aren't part of the admin group if we look here at Craig Denton at Craig Denton's details he's just a user he's not part of the admin role so this should we should get a 401 unauthorized error so let's see if that works let's see if Craig Denton is prevented from adding a review to the database by our code okay so let's hit the send button 404 not found okay so that's not what we wanted review update review what okay so that's not the error this should be a patch that's why we've received this 404 not found so let's try again it should be a 401 okay so not a 401 we've actually sent back a 400 bad request and that's just because that's the way that I wrote the code so if we go back to the code that is correct we have been kicked out we weren't able to use the update review functionality but I think this would be more appropriate as a 401 so this should be status unauthorized yeah you don't have the authority to access this endpoint let's save that so let's save that and let's test that again so go run dot enter okay it's listening on port 8080 let's launch Postman and let's try that again okay we've still got our settings there last time it sent us back a 400 bad request that's because of the way we wrote the code the logic is correct but we should be passing back a 401 unauthorized exception rather than a 400 bad request because this user is not part of the admin role it's a valid user Craig Denton that can be authenticated but not as part of the admin role and only administrators can add reviews to the system so we want Craig Denton kicked out when he tries to access uh the update review endpoint so let's press the send button 401 unauthorized user must be part of the admin role great and that's pretty much the server side code all written and we can now move on to writing the client code the React code so we've coded the server side part of our code and have successfully tested each aspect i.e each of our endpoints successfully hopefully you've noticed that at times during the code creation process I have flagged in the way of including arrows and appropriate text in post-production that even though the code works there is a better way of writing the relevant code so this is what we are going to cover in this section of the course we are going to look at some practical examples of a few go jinggonic best practices by appropriately amending just a few pieces of code that we have already created and successfully tested so we are going to address a few minor coding issues in this part of the course we are not fixing any code per se but we are amending the code in order to adhere to best practices please note that any keys for example secret keys like these ones here or API keys like for example open AI API key here for interacting with endpoints on open AI i.e used for accessing third-party endpoints for security reasons can come from a key management system from a cloud provider and therefore in production would not be embedded within the ENV file as we are doing here we are reading in the values from thev file here let's look at our code for example you can see here we're checking to see if that env file exists because we're running it on our local machines here but regardless of where we're actually reading the relevant environment variables from this line of code is actually reading in the value and this could come from a key management system or it could come from the cloud provider's own environment variable facility environment variable management facility um so that code doesn't actually change but here we are just checking whether the env file exists and we're not throwing a fatal error we're not logging a fatal error if the env file doesn't exist so when the code is actually running in the cloud this will actually result in an error occurring because thev file will not exist on the cloud but all it'll do is log a warning it won't log a fatal error and this line of code here gets called regardless of whether the code is running in the cloud or locally so it doesn't actually matter where the environment variable is coming from but we are performing a check here just to make sure that it is on our local machines so that it can tell us that our code can tell us as it were if the environment variable exists on our local machines um or not and ob obviously when we're running it locally the environment variables thev file must exist on our local machines but it won't exist in the cloud so this will be logged in the cloud which is fine because it's just a warning message it's not a fatal we're not logging a fatal error so it is best to store the relevant key values so we got these key key values here for example the well we've got this key value here OpenAI API key and these uh secret key values here that sign the relevant tokens it's best to store those in the relevant cloud hosts key management system as a security measure as an extra security measure right so if we go to let's perform our first best practice code amendment if you like so let's go to database_connection.go here so here we are handling our connection to our MongoDB database we also have a reusable function called open collection here which is reused throughout our application where appropriate code is executed to open a specified collection within our MongoDB database so we are using the MongoDB for go driver here within our code to interact with our MongoDB database and we are centralizing this reusable functionality within this file now here's the actual issue when we import this package what we're doing here which is not really good is we are actually automatically dialing the MongoDB uh instance so we we're dialing MongoDB here when we import the package we actually don't want our code to automatically dial MongoDB we want it to only establish a connection so by dialing I mean establishing a connection with MongoDB we only want to do that at when when the MongoDB connection is actually uh necessary when it when it's going to be used so we don't want to do this uh like like we've got it here currently so we're going to copy this to the main method we're going to call this code from the main method and pass in we're going to inject the client into the where we're setting up the roots here and then we'll inject it into the relevant handler function methods so that's what we're going to do but before we do that this DB instance isn't a really good name for our method so because we're using the DB prefix here which has special meaning for MongoDB so instead of calling it DB instance I'm going to change that to connect so we'll keep the method name very simple and just call this connect like that and then we can change this code over here to connect but we're going to have to change this code further because we're going to import this database package from within the main package and then this code will be changed to database.connect but let's cut this code ctrl X and I'm going to go to the main method now here and I'm going to paste it in before this line of code here uh sorry after this line of code here just before where we're setting up our roots I'm going to paste in this code where we will connect to establish a connection with MongoDB so we need to import the database so let's go back to database connection there and we're also going to want to import these we're going to want to import this package here so let's go back to main.go and make sure that we're importing that package so let's ensure that we're importing the mongo package there okay so that underline has gone away now but the problem here is we haven't imported the database package so let's go here where we're importing the database package and just copy that importation code so this code here and let's paste that here like this and then we can call is there a problem there uh yep let's paste that in it's because I saved it and we're not using it it removed this line here so I won't save it but we want this um importation code in place because we're going to use the database we're going to use database.connect here dot connect like that and now we're going to pass in the client to where we're setting up the roots here and in fact this should be a lowerase C for client because it doesn't need to be available to any calling code so we'll keep this a local variable client and the underlying CR because we haven't actually included a parameter definition for these client values here so that's the next thing we need to do so if we go to setup setup protected roots protected_roots file.go we now want to include the client here so client which is of type mongo.client like that okay and we don't have the mongo package so we need to paste in the mongo package where we we're pasting it in here so we need to paste that in within the roots package also so let's include that there within the sorry protected_roots.go file so that we have access to the relevant type there and then we're going to also inject this client into all of these roots here these handler functions these genonic root handler functions if you like endpoint handler functions they need the client in order to connect connect to MongoDB okay and we need to do the same in the unprotected roots so let's do the same here and we need we can just paste in this package importation code here like that and then we need to include this parameter definition within the relevant method like this and then we of course have imported that so that's working correctly and then we want to just pass in the client like this and of course you've noticed that there's red squiggly lines there because we don't we haven't included a parameter definition within these handler function these jonic handler function methods so um we'll do that in just a bit let's just copy let's just paste the client in like this so that it is being passed in to these handler functions here these gingonic handler functions so these functions handle the logic for when these end points are called via the relevant HTTP methods get post get and post in this particular case so the next step as you can see is to include the client this definition the client definition for parameters within each of these handler functions let's handle this one first so movie imdb so it's get movie here so get movie we can actually just navigate to it by selecting the method right clicking and go to definition and let's include that definition like that there great let's go back there and now add movie we need to do the same here so go to definition and let's paste that in there and we just repeat it you can see the red squiggly lines are disappearing as we make the relevant coding amendments so go there and paste this definition in there let's go back to protected roots there and we want admin review update we need to amend that code and let's do that there so we're passing in the client there okay and let's do the same now for for the the roots for the protected roots the the sorry the unprotected root handler methods if you like the function the handler function methods here so let's go to get movies and let's include that definition there and let's just go down and repeat this process for each of these handler functions like this okay get genres okay and then refresh token handler great and then let's just go through each of these functions and make sure that our code is amended appropriately okay so next thing I need to point out is these variables we actually don't want to do that here um it's quite cheap to create these connections to or open these collections it's quite cheaply done behind the scenes so we can actually just open these collections as we go so I'm going to cut this code here and I'm going to include it here just above this line of code here like that and now we need to pass in the client like that we're going to inject in the client like this for this open collection method and now we need to amend this open collection method so let's go to do definition here and we need to include this definition here like that because we're going to be passing in the client like this here and let's make this a small C where we're connecting to the relevant collection like that so we're opening the collections in a slightly different way now and we've amended our code there is this reusable code which is centralized within the database_connection.go file here and then we can just go back to moviecontroller here and that should be pretty good that's okay and I think that's all we need to do in regards to opening the relevant collection okay and then let's go to the next method so let's go to protected roots we got add movie here so let's go to the definition and we want to do the same thing okay so we actually amended get movies there um we want to do the same for get movie okay so it's the same kind of code we can just paste that in there like that we're passing in the client and we're handling the movie we're opening the movie collection in a slightly different in a slightly different way now like that okay so let's go back to protected roots there so we've done ad movie let's make sure that we well we have amended ad movie okay we haven't so we need to do the same thing here so we can open the collection here like this for the add movie method okay let's go back to protected roots so we've done get movie add movie and we want to do re get recommended movie get recommended movie sorry so let's go to definition there and it's just the same repetitive task of making sure that we are now opening the collection appropriately let's go back to protected roots here and let's deal with admin review update so let's go to definition there and where we are making our colle our connection where we are opening our movie collection we need to include this line of code here just before our code engages with the movie collection here great okay so we're now handling our movie collection connecting to our movie collection appropriately for the protected routes so no red squiggly lines here we're getting there slowly it's a bit of a tedious process but it is worth it okay and then we've got get movies and we actually I inadvertently did that thinking it was get movie so we've actually amended this code here already and all we're doing is opening the movie collection before we're using it we're opening it locally here not up here we don't need to do it up here anymore and we're going to do the same for all of these collections in just a bit so we're now opening these collections in a different way locally when we need them and we're doing that because it's fairly cheap to open these um collections behind the scenes so it's not a problem to do it this way okay so let's go to So we've handled that unprotected roots we've handled get movies register user so let's go to definition and of course now we're changing the code regarding this user collection so let's cut that and we want to paste it in here just before we're using the user collection like that there and that's all we need to do and that's not all we need to do because we haven't passed in the client we need to pass in the client like this and then let's make a copy of that code let's go back to the unprotected roots code here and login user so we're going back to the same controller and amending login user with the way we are now engaging or now with it the way we are now opening the relevant collections and we're just pasting that code in there so that we're opening the user collection appropriately before we use it within the login user method and then let's go to unprotected roots again log out will be the same thing so let's go to definition log out here okay and with log out we're updating all tokens so we'll have to pass in the client here to this update all tokens method within utils so let's include the client there because we're now passing in the client to each of these handler functions so we include client there and then we need to um we need to copy this definition here so that we can update the update all tokens method within our utils package and we need to include this definition here client MongoDB and we're using the now the user collection at this point here so we need to amend we need to include a line above this code that actually makes that connects to the colle that opens the collection okay so here it is here so we can just cut this code here and paste it in just above the line of code where we're actually using the collection of course we need to pass in the client like this okay let's go back to unprotected roots code the setup unprotected roots function and now we're dealing with the genres here so let's go to that method go to definition so you can see here we're using the genre collection so we want to eliminate this code here that we've defined outside of any particular function we don't want to do it like that anymore we want to do it the way we're handling the get movies collect the opening of the movies collection so we need to go back to sorry this unprotected route get genres handler function and paste that open collection code just above where we're using the genre collection and we want to pass in the client like this okay and that is looking pretty good so we're getting there let's go back to unprotected roots and then we got refresh token handler i don't know if we're actually interacting with the database in this code here validate refresh token yeah we are we we've got the we're engaging with the user collection here so we need to update this method with a with code to open the user collection here okay so we can just copy the relevant code from uh yeah let's go back to unprotected roots refresh token handler go to there and then just above here include as I said earlier if you get stuck at any point and you're not able to follow along with this code or you find that you get lost um you can check out the final code at this URL the URL to the relevant GitHub repository has been included below in the description okay so we're updating it there find one right and then we've got update all tokens we need to pass in the client here excellent that looks good and then I saw a red squiggly line up here there it is refresh token update all tokens we just need to pass in the client when we're logging in the user we're updating the tokens within the user collection for the relevant user so we need to pass in the client object here that connects to the to MongoDB okay so that's the connection to MongoDB there and we're now handling opening the relevant collections as we need them within local function handlers or handler functions okay so one thing we haven't done is we haven't eliminated if we go here we are still using this rankings we're still opening the ranking collection in the old way and this of course is now not going to work so what we want to do is cut this code here and we need to find the function called get ranking now it's this is a method that's called from within one of the handler functions so it's not being called directly by an end an end point so we need to actually include the client definition here so let's do that client uh star mongo dot client like this and then let's include the code that we just copied to our clipboards that we cut and let's paste it here and then let's pass in the client object like that there so now we're opening the collection in the in the in a consistent way and we're doing it now for the rankings collection so we're only opening the collections when needed within the local within the various local functions okay great and then we need to find out where this has been called so we can go find all references go here and we need to Okay we're going to need to pass in the client down to here also so I'm going to include client as star mongo.client like that and we need to pass in client there and then we need to find where we're calling this so we can find all references there there we go and here we're defining client within the ad this is the actual handler function the jinggonic handler function so we're handling an actual endpoint in this function here and we're calling these these functions from within the parent handler function so we're calling get review ranking which calls get ranking and we need to pass in the client here so let's client there and the red squiggly goes away and now we are handling opening the relevant rankings collection in an appropriate way if you like okay we've got a problem there let's go and see what the problem is undefined user collection here so we've so this is also a function that's being called by one of the handler functions um and we don't have the client object available to us we're not opening the user collection in an appropriate way yet here so we need to handle all of those things so we need to find where we are opening the user collection or users collection and let's go back there and let's go to where the problem is and let's paste in that code now we're opening we're opening the collection appropriately but we don't have the client object so we need to include the relevant definition for the client here the MongoDB client so client and that's defined as mongo.client like this okay and we're passing it in as an argument there like that okay we still have a problem okay yep let's go to the problem and of course we this is the get recommended movies handler function which handles an actual endpoint call this is the logic for an actual call to an endpoint using a HTTP method from the client so we need to pass in the client that is passed into this handler function to this method like that we have no problems now excellent so we're also opening the rankings collection appropriately now so what we're doing here now is we we're only opening the collections when we need them from within the relevant functions whereas before we were opening the collections outside of any one particular function call which is not what we want to do so this is a better way of doing it excellent okay so the next thing we want to do and I would say this is actually quite an important fix is we want to pass in this ginonic context object to the width timeout method here and let me explain why we want to do this okay so this is quite an important fix we are currently not handling this resource clearing related code correctly within our Jonic application here we are passing in the context.background object to the with timeout function we should be passing in the jinic context here jin implements context in a way that it embeds the requests context.ext so you can safely use it wherever a context.ontext is required why not context background like we're doing here if you use context.background background instead you lose request cancellation propagation if the client cancels the HTTP request closes the connection navigates away etc jyn cancels the jin context so using C as the parent context means your timeout context is automatically cancelled when the request is canled using context.background background ignores client cancellation so your handler may continue doing work even after the client has gone away that can waste resources you lose deadlines and other request scoped values jyn may attach request scoped metadata or timeouts in its context using C ensures you inherit them use C your context respects client cancellations and request scoped values use context.background your context is independent ignores client cancellations and could lead to resource leaks so we must pass in C the gingonic context here rather than context right so let's handle this let's make sure we are doing this so let's pass in C here firstly okay and so that's basically what the fix is it's very basic so we need to do it wherever we see this we need to pass in the gingonic context instead so let's select that and let's go to find all references and let's go through them one by one we've handled that there and we need to handle it here so we uh pass and see there go there pass and see here okay select this one so we need to pass in C here for admin review update there so we're passing in the ginonic context object that is passed into this anonymous function here this is where our root handling functionality exists this is the handler function for an endpoint so we're passing in the jin context object instead of context.background background um for the resource handling code here okay excellent let's go to the next one okay passing C here next one pass in C here okay go to the next one we need to pass in C here now this one's a little bit different because this is called from an actual handler function so we're not implementing a gingonic handler function for this method so we need to pass in C to this method so C so we can do that at the end here let's go comma C and let's define it appropriately as sorry Jin dot context like that there and we of course need to ensure that we're passing the context to this um to this function here so let's go find all references and we need to pass in C here to fix that there okay brilliant find all references and you can see we are here this one here we're not yet this is uh this is calling get rankings from another function that's not a gingonic handler function so we need to also pass in the C here the Gonic context so let's do that c uh as star jin context like that and then we can pass C down there to get rankings function and then we can just find all references here and make sure that we're passing C in here like that and that looks pretty good we're now handling this clearup code this resource clearup code effectively by passing in the gingonic context to the width timeout method here which clears up the relevant MongoDB resources so the next code I want to amend is within the main.go file the main package here so it's actually good practice to ping the database before we set up the roots here because if we don't have any connection to the database then our code is going to fail so this is just a better way of handling the case where the where we are unable to establish a connection to MongoDB so we can do that with this code if equals to client we've got our client object now we're connecting to we're estab trying to establish our connection here and then we're going to try and ping we're going to try and see if we can if our mong if MongoDB is receptive to us connecting to it so uh this is not connect context do background like this and then null we don't have jin gonic context at this point so we can use context.background background as the context here and then null and then on the same line we can check the error to see if this object has a value so if is not equal to null okay we want to handle an error if we are unable to ping of course null has one L if we are unable to ping MongoDB we actually want to log a fatal error here so we use fatal LF like that and we can include an appropriate message here failed we're not reaching the server failed to reach server so we're flagging a fundamental issue here so we don't want the code to continue there's a fundamental issue and by logging it we we should by examining the logs understand what is actually happening so there's a problem with us connecting or reaching um the actual server so our client can't ping MongoDB so we need to actually stop execution here and log an appropriate error and it's a fatal error so we don't want the code to continue and set up the roots because we know that it's going to fail downstream anyway so we can handle that issue at this point here in the code within the main function there and the other thing I want to do is um make sure that we're disconnecting from the database and we can ensure that we disconnect at an appropriate time from the database and this is to do with clearing up resources so defer funk and then we want to equals to client disconnect like this disconnect okay we need to pass in the context dot background method there like that and then we can check our error object and see if it's null if it's not null we want to log a fatal error here okay and we can now let's amend this message so that it is appropriate to what is occurring here so we're trying to disconnect at an appropriate time we're trying to disconnect from MongoDB say for example an error occurs we want to disconnect from MongoDB and if that fails we need to log that fact so that we're aware of what has happened so failed to disconnect from MongoDB like that okay and we're logging the appropriate error open and close brackets there space there and that's pretty much it so we're just making sure that these cases are being handled appropriately making sure that we can connect to the database to MongoDB before we set up the relevant routes so that it fails here at this point um before it's going to fail anyway if the if we're unable to connect if we're unable to reach the server at this point here so that's what this ping functionality is all about and then this defer functionality is just to make sure that we're disconnected from MongoDB we don't want to have those resources hanging about in memory unnecessarily we need to make sure that we're disconnected from MongoDB okay excellent and the other thing is I'm not currently checking whether the environment thev file exists before calling the code to read in the appropriate environment variable value here and on our local machines that's quite important so let's include the relevant code um so it's just colon equals go do go.env.load and then within quotations env like this and then we just check our error object if it's not equal to null an error has occurred and we just want to log a warning at this point because this error is going to occur when it's deployed to the cloud but locally this um warning message can be helpful for us because then we know we haven't created thev file and we need to do that and this is the reason why certain code is failing so at this point we're checking to see if that.env env file exists before trying to read one of the environment variables here and this code doesn't change whether it's running locally or in the cloud it's going to read the relevant environment variable in the same way regardless of how it is stored and it will be stored differently in the cloud to how it is stored locally in thev file here but we need to just check whether that file exists and it won't exist in the cloud so this it will just log a warning message in that particular case it won't be a fatal it won't log a fatal error it's logging a warning message unable to find file like that okay excellent okay so now it's time to create a front end for our application and we are going to use React for this purpose you can see here we've uh created a client folder so our serverside code is within the server folder within the root folder the magic stream movies root folder we've got a server folder and we've got a client folder so within the client folder we're going to create the react code within the server folder we've got the Go Jonic MongoDB related code here we are going to create a React project so we are going to use a technology called Vit for creating our React project vit has an advantage over traditional build tools like create react app and bundlers like Webpack for example it offers faster dev startup and rebuild time simpler configuration and out ofthe-box simpler defaults so we're now going to use VIT to create our React project to use Vit you must have NodeJS version 14.8 plus 16 plus or newer we also need npm which comes with Node.js JS and is used for installing dependencies so if you don't have NodeJS installed navigate to NodeJS.org and you can download and install the latest version of Node from here excellent and you can check what version of of Node you've got installed simply by typing node dash V like this and you can see version 2401 that's my version of node and if you want to check what version of npm you've got type npm slash sorry -v like that excellent so once you've got that set up i.e uh an appropriate version of node and an appropriate version of npm we're ready to go so firstly we need to be in the client folder so let's type cd client because we're going to be creating our client code within the client folder and we're about to create a react uh project using vit so to create the react project using vit we then type n pm create like this vit and then at latest like that okay and then to proceed we press Y so it's telling me that I need V 7.1.0 need to install the following packages so I'm going to say Y here so we need to give the project a name i'm going to call my project magic stream client like this press the enter key excellent and then you can use your down arrows and you want to select React here press the enter key and then select JavaScript from this menu here and now it's instructing us to type the command CD to navigate within our Magic Stream project the root of our React project and then we want to type npm install which will install all the necessary dependencies to run a react application and then we can test our project by typing npm rundev so let's follow those instructions so let's go into magic stream client stream client like that okay okay and then we want to install all the relevant packages within the magic-stream-client folder so to do that we can use npm so type npm install press the enter key and it will install all the necessary dependencies that we need in order to run our React project as always we just need to be patient while it installs those dependencies on our local machines excellent that wasn't too painful and that's all done okay found zero vulnerabilities that's good let's clear the screen and now we can we should be able to run that so you can see here it's created our folder Magic Stream Client it's created the node modules folder which has all our React dependencies great so we've created a basic infrastructure for our React application using Vit and there's a lot of advantages to using Vit over the traditional create react app facility right okay so we've now got the infrastructure for our React project set up so vit has set up a basic project infrastructure so there should be some very basic functionality included in that and we can test this by typing npm rundev like this okay brilliant so it's listening at port 5173 on localhost so let's copy that URL let's go into our favorite browser in my case that's Chrome and let's test that the infrastructure has indeed been created for our React project excellent look at that and there we go we can test the interactive functionality by pressing this count button and it counts within our browser as we click the button it creates a it increments the value within this button we can see that our user interactive functionality is in place and we can start developing our client application now that's brilliant so make sure you get to this point before you start developing the client side for our magic stream application excellent so let's go out of that and let's go back here and we can press Ctrl C to cancel out of this so it won't be listening on this port on our local host so let's press Ctrl + C and let's clear the screen we're going to get started with an easy win and we're going to start with a functionality that just displays kind of a gallery of movie posters on the front end so to do that we're firstly going to create a folder within here which will house all our components so new folder and I'm going to call this components so we're within this src directory which was created by VIT by default we're creating a components folder here so we got our components folder and we're going to create our first component within a folder named movie so our first component will be a reusable component that will represent one movie so it'll be represent basically a front end a card which will contain the poster of the relevant movies so we're going to create one movie one movie component that can be reused for all of the movies throughout the application so that's what we're going to start with okay so let's create our movie component and we're going to create that in the form of a JSX file so let's select the movie folder here press this icon for new file and create a file called movie.jsx so this is a JSX file this should have a capital letter so let's stick to conventions here and let's make sure that the movie file name has a capital M here okay great so we're going to be using Bootstrap as our CSS framework for styling and layout purposes and an easy way to utilize or leverage Bootstrap is through a package called React Bootstrap so we want to install React Bootstrap so we must make sure that we're in the magic-stream-client folder so to install React Bootstrap we type npm and then I or install let's type install and then react Bootstrap like this excellent and we've got React Bootstrap installed so let's start building our movie.jsx jsx component our movie component within the movie.jsx file okay great so I'm going to start by creating the infrastructure for the movie component so just const and then movie with a capital letter like this and then equals to and then this is where we'll pass in our props and we're going to pass in a movie prop to this particular component and then let's implement the structure for our component here so this is the the basic structure for our movie component done here and then we can return the relevant JSX code to the calling code like this and then below here we want to export export the component as default so that the calling client can easily import this component and we'll look at how to do that in just a bit great and now we simply write the code the JSX part of the code for the movie component firstly we need to import a few things so in fact at at present we just need to import one package and that's the button package from React Bootstrap and we can do that with this code here so React Bootstrap slash button like this but there's a few other little steps that we need to carry out in order to be able to leverage Bootstrap the way we want to within this component and other components so so we need to go to the main.jsx file here there it is main.jsx file and include this line of code import within quotation marks bootstrap slashdist/css slash bootstrap domins like that and this will enable us to leverage Bootstrap some of the Bootstrap classes from within the components of our application so let's go back to our movie.jsx file and implement the code for this so we first want to create a div and I don't want to focus too much on the styling code because it just take too long to explain every class that I'm including here for styling purposes or layout purposes so I would urge you to look up these classes on the Bootstrap 5 website or the React Bootstrap website if you want to know more about the details of these classes okay and it's kicking up a fuss because I've included a dash here so we need to include an equals oops an equals there great that looks a lot better okay and let's proceed so this movie by the way this is what this movie prop will be the object that contains all the data that we'll retrieve from the server and we'll do that a bit later so just take it for granted that this movie data has been returned from the server and we're now able to leverage its various fields within our movie component okay so let's create another let's create a nested div here class name equals 2 and then let's include our class names here card and these are all our Bootstrap class names that are used for styling and also for layout purposes within our movie within our movie component here okay okay let's include another nested div here okay so let's go class name like this equals to and then card body these are bootstrap these are standard Bootstrap classes that we're using to style flex deflex and we're using flex grid so we're leveraging the flex grid technology through Bootstrap here for our card and within our card we're displaying all the relevant information about the movie so H5 let's include the H5 element here class name equals to card title so this is where we're going to include the card title and then this is how we can display variables within and mesh those variables within the HTML code and this is just JSX so it's not actually HTML code but it represents HTML code if you like and then we can wrap any variable data that we want to display on the front end within curly braces like this so movie dot title like this so we're displaying the movie title within the card within this H5 element here okay okay and below that let's include a P tag like this type class name equals to and then let's include an appropriate Bootstrap class so this is just for styling purposes you'll see the significance of this styling when we run the code once we've finished some basic code here to display the movies on the front end okay so we got card text and MB2 that's just margin bottom two that's what that represents and within curly braces let's include moo v dot and we're going to include the IMDb ID within this element here so we got the title and the IMDb ID displayed so far and this is within the body of the card that will contain all the movie data okay below this let's include the ranking value this is the sentiment that we discussed earlier when we were creating the server side code so if ranking ranking name So if ranking name is truthy let's include this code here to display the ranking so let's include a span element like this okay and a class name class name equals to and this is just more Bootstrap classes so we're using the badge class and this is just for styling purposes makes us makes it easy for us to style our code without having to go into the details of the actual CSS and let's include this with a font size this is just our own custom CSS so that we can override whatever font size is there at the moment and we'll include and this needs to be within further curly braces so we need to wrap this in two curly braces because we're including an object so font size and then let's include the font size which is just one rim here okay okay and then within curly braces let's include the ranking name which is just the sentiment of the movie you know excellent good okay bad or terrible ranking name like that okay brilliant so we slowly building up our movie component and then up here we want to actually include the poster image now this is obviously the most important part of the display for the movie the movie data the poster image because without the poster image you're just going to have a boring title and the IMDb value which isn't particularly visually appealing so let's include the poster image here so to do that I'm going to create a div here okay div and I'm actually going to include my own custom inline style here so style equals and then open curly braces and then let's include an object position uh relative here so this is just inline styling here and this is how you do that you create an object to represent your inline style within the JSX element like this so position relative and you'll see the outcome when we test the the code in just a bit but for now let's just create the component so we want to include the poster image here and then for the source we want to include a variable here so we don't want those in quotes so it's src equals to movie dot poster path and that's the poster path that we want displayed for the movie on the front end okay brilliant and then for the alt we'll just include the title like this movie dot title like that and let's include a class name like this so card image top like that and then our own custom styling here so let's include our CSS code here so object fit contain like this and I'm assuming you've got some knowledge of CSS it would take too long to explain all of these properties these CSS properties 50 but please by all means look them up very important if you're a web developer to understand at least basic CSS percent right okay so I think that's okay for now for our component it probably isn't perfect but hopefully it'll display all the relevant data on the front end okay so within the components folder here let's create a new folder called movies so this the folders should be just to stick to convention the folders should have a small letter in front of the folder name like this at the beginning of the folder name so the first character should not be capitalized so we got movies and then within movies let's create a file called movies.jsx like that so let's create the code for our movies component so firstly we want to import the movie component and we can do that like this from and we go dot dot because we're going back a directory and then you can see in sense has suggested the folder so movie and then the component is movie like that okay okay movie is declared but it's not used okay so let's create the infrastructure for our component so const movies equals to and then we pass our props through like this and we want movies so that will be a collection of movie data that we've that we will in a bit retrieve from the server so we'll contact the movies endpoint and bring back the data and then this data is passed into our component by the calling client which we'll create in just a bit so this one we want to include a movies prop and a message prop here and then let's create this as an arrow function like this and let's return the appropriate JSX for our movies component and once again I'm not going to go into detail about all the the various Bootstrap classes that I'm using here so class name equals to container so this is just a bootstrap class and margin top four this is just wraps or abstracts away the CSS code for the margin top setting let's create another div here and I'm just going to set the class name to the class named row and this is for layout purposes so we're using the Bootstrap grid system for layout purposes here and let's open curly braces like this movies we want to check that movies contains a value or is truthy and movies.length so there is indeed data within the movies collection being passed as a prop to this component we can do that with this code movies.length is greater than zero like this so if this condition is true go question mark and we can handle the true condition like this if the condition returns true we want to use movies the collection the array dom map so we're using the JavaScript map function to loop through the movie data within our movies collection and this is how we can do that so we go movie like this and we use an arrow function to establish our logic so so if movies is truly and the movies collection contains movie data we're looping through the movies collection and we are going to display our movie which we've just created within the movie component and we need to include the key attribute here and set it to a unique value and we can do that through this code we can set the unique value to underscore the underscore ID property within our movie data bear in mind of course this data is coming from the MongoDB database and we've got this ID underscore ID value established within the movies collection to uniquely identify each movie and we're using this as our key for our movie element here so this helps react identify each unique element okay and then we go movie we want to pass down the prop movie to the movie component so we go movie equals to within curly braces then the movie data gets passed down as a prop to the movie element like this okay we can just Close this element like that get rid of that excellent okay and we have a little issue here let's just check we'll check that out in a bit let's just finish off our if else condition here so if that's true we want to display a movie we're looping through all of the movies so if the movies collection contains data we're looping through all of the movies within the collection and displaying the movie on the front end appropriately there else here so the colon signifies the else condition and then we can just include the message that will be passed down to this component as a prop within this H2 element like this okay H2 and then within curly braces message like that okay let's just take care of this issue here so let's look at what's going on here so we've got our movie component we're exporting it as default here movie doesn't look like there's anything wrong there we've got it components movie.jsx and then within movies okay I'm just going to write the export functionality here so we want to export this as default export default movies like that okay brilliant and now this is obviously worrying me here uh I know what actually is going on here it's because it's actually painful when you do this in React and it complains it's because I initially um created this file with a with a small M at the beginning of the word movie and I think the this is why it's flagging this as erroneous because then when I changed it it's not picking up the change for some reason when I changed it and capitalized this M it's not picking up the change so this is a bit painful but to get around that um I'm just going to cut that cut that code put it in my clipboard i'm going to delete this file i know this is a bit of a pain move to recycle bin and I'm going to create the file again new file this time with movie.jsx like that and I'm going to paste that code that I just cut into this new file and save that and let's see if that's that has resolved it yeah so for some reason I get an issue when I first when when I change a file name it doesn't seem to be to propagate through and if you initially didn't get the file name right it can complain when you're trying to import a particular JSX component within another component and that's what was happening there so that's a bit painful but that's resolved now okay so uh and then we want this displayed on the homepage so I'm going to create a component called home we want the movies displayed on the homepage so I'm creating a folder called home and let's create a folder called home like this with the name of the folder the first character of the name of the folder will be in lowerase here so we're sticking to that convention and then let's not make the same mistake the actual JSX file will now have a capital letter for the first character of its name so home.jsx and we're now going to create the code for the home.jsx component or file right so screen there right so let's uh do that so now within this component we're actually going to be calling the serverside code so before we start writing the actual code for the home.jsx file I'm going to create a folder within src called API and this will house the code for connecting to the server just the basic code for that i'm going to create a new file called aios config we're going to use the package the axios package the axios dependency to handle the infrastructure for calling the server side code so we actually need to install Axios first before we configure of of course before we configure Axios yeah to install Axios we simply type npm here at our terminal prompt npm install aios like this brilliant so we've got Axios installed and we can now write the just the infrastructural code the configuration code for using Axios to call our server side endpoints so let's go Axios import Axios from oops Axios like this okay and we want to read in a base address so it's going to be hosted on local host initially but we want to be able to configure the base address for where our serverside code is running so that we can easily change this value when we eventually deploy our code so let's create an environment file aenv file on the same level as this src directory so we want to actually create that within magic stream client here so let's create our env file like this env and let's configure an environment variable and you need to adhere to a naming convention when you have created your react application using vit so to do that we have to include vit like this and then our the name of our environment variable so I'm going to call this API base URL okay and we're going to initially be running our server side code at local host colon port 8080 like that so we've configured our environment variable let's go back to Axio's config here and let's read in that environment variable so API URL URL like this equals to import and this is how we read an environment variable using VIT because we're using VIT we've used VIT to create our project so we have to adhere to certain VIT conventions api underscore base URL like that okay we can include colons here okay so we're reading in the base address for our server side endpoint here and then let's configure Axios so export default Axios dotcreate let's open the brackets there and then let's configure Axio so the base URL property must be set like this base URL API URL and that's of course the the value that we've read in from thev file so it's the environment variable and then within the header we want to include this configuration here headers and this is an object JSON object content dash type colon and then application for/json like that we didn't close this quotation here and that's why it's complaining okay brilliant so we've got Axios configured now great so now let's go back to the home component so firstly I want to import these commonly used hooks for React so use state use effect from React whoops okay then I'm going to import the Axios client so we're importing the component that we created it just earlier from the appropriate directory API so it's in the API directory and then it's picked up that it's Axios config so we're importing Axios config component here like that great so we also want to import the movies component from this directory movies whoops dot movies with a capital letter and let's create the infrastructure for our components a const home and we do that in the form of an arrow function okay and then we include the arrow open curly braces okay and we want to create state for a movies variable so state variable and then along with the state variable we've got the set movies function which sets or changes the state of the movies state variable and we go equal to use state and we pass in an empty array by default so no movies yet but we want to populate this movies state variable with an array of movies a collection of movies which we will retrieve from the appropriate endpoint on the server okay okay and then we want to create a loading state variable set loading like this equals to use state and then by default we want this to be false so it's a boolean state variable and then a message if in case an error occurs we need to display a message to the screen set message equals to use state and now we're going to use the use effect hook use effect hook so that when this component loads we call the movies endpoint on the server and populate populate the movies collection with the appropriate movies data that we retrieved from the server okay let's go const like this equals to we want this to be called asynchronously so we use the async keyword here and we include another arrow function like this open curly braces and within this we create our logic for calling the movies endpoint so and we're going to use axios to call the movies endpoint so we set loading equals to true because we're about to call the serverside code and we want to display a loading indicator on the front end while the serverside code is retrieving the relevant data and returning it to us uh let's set this message to nothing here so we're initializing the message to an empty string here let's include a try catch block here so that we can handle errors if they occur okay there's that and a finally block here which gets called whether an error occurs or not and here we can use this finally block to set the loading boolean value to false because we want the loading indicator to no longer be displayed whether an error occurs or not so we can include this code within the finally block and set the loading state variable to false here so we've set it to true up here and we're about now and now within the try block we're going to call the relevant endpoint to get all the movies data so we're using Axios client and we go get and this is how we call our endpoint on the server just for/mov which is our endpoint that will return the relevant movies collection to our client React code and then let's set the movies state variable using the set movies function to response data which should contain the relevant array of movies okay and then if response do data so if an empty array of data is so no movies are found on the server equals to zero then let's set the message to set the message of the message state variable to an appropriate message so there are currently no movies available so if the say the database is set up but the movies haven't been imported or no movies have been added to the relevant collection in the MongoDB database this is the an appropriate message to display on the client okay and let's handle the error we're just going to console log the error here like this okay error fetching movies will be good enough I think so error oops error fetching movies go like this and then comma error like that and then set the message to set the message state variable to an appropriate error message and this is the same basic error message just error fetching movies we've included a bit more detail in the console window if an if this if an error occurs at this point if an error occurs when we're trying to use Axios to fetch the movies from the server okay and that is our use effect hook written so this hooks into a React life cycle so when the component loads we're sending off a call an asynchronous call using Axios and hopefully the endpoint returns us valid movie data here and we're setting that state variable m movies here appropriately we need to pass an empty array like this to the use effect hook so that means that when this when this when the component loads this code fires and populates hopefully populates the movies collection with the movies data so that we can now return JSX to to the calling client okay so let's include empty tags here and then if the loading state variable is true we want to just display a loading message and we can just do that by returning say H2 we'll include a proper spinner later on as a loading indicator but for now let's just include the text loading dot dot dot and then else condition fires if the else condition fires here we actually want to display a gallery of movie posters if you like the movie data movies equals to and then the retrieved movies within the state variable the movie state variable should be retrieved and then if there's a message also passed down the message state variable as a prop to the movies component we have a few problems here okay what's going on okay we need opening bracket here we need an opening bracket here and a closing round bracket here to wrap the else part of the if statement okay so I'm not currently closing this movies element and that's why we're getting all these red squiggies so use a closing tag there and now our code looks good so within this fragment here which just means that uh this is a substitute for an element but we don't want a particular element here like a div or p element for example so we can this is a way that we can include a parent element without including a particular HTML element for the parent element so we're including a react fragment here so just we can do that by implementing empty tags like this to house JSX elements here so if so while the data is being loaded while the data is being retrieved from the server this loading indicator should display or else we want the movies displayed to the user okay so now we've got all these components set up it's still complaining i don't know why it's giving us the same problem as we had before but I I think I seem to remember that I got it right with this one okay so this has got a small letter this should have a capital letter here so I'm probably going to have to do the same thing i'm going to copy all of this code and cut it and delete this file here sorry about this and then create the movies JSX file with a capital letter at the beginning jsx and then just paste that in shouldn't really have to do that but So do the same thing here just to fix this ridiculous problem delete that there paste that in there okay and that should fix everything so I don't know why that seems to cause a lot of problems if you initially get the file naming convention incorrect and you want to change subsequently change it it seems to cause havoc it seems to flag a lot of associated errors so it's best to get this correct from the beginning I guess but um to get around it you can just copy the code to your clipboard delete the relevant file and create a new file and make sure that the first letter is capitalized so okay great so that's done so I think the next step I think we can now that our code looks good we can go to the app.jsx code here and one thing I actually want to do is just get rid of all of this here okay I'm going to just get rid of the the app.css code so it doesn't interfere with our CSS code so app.css and the other one is index.css i'm just going to delete all of that and then we can go to the app.jsx file here and we want to replace all of this code here now with just a call to our home component okay we don't need all this here and I'm going to import the home component so I must make sure I'm exporting it first so oops so type export default default home like that okay so export default home so firstly we want to import the home component within the app component so import home like this from comp from then within quotations dot slash components slashhome slashhome like that and we're importing the home component and then we can just include the home element here within our code for now just to test whether we're able to call the movies endpoint retrieve the data and then display the relevant movie gallery if you like to on the front end okay so we want the server side code now running because we're going to contact the endpoints running on our local machines so it's going to be localhost colon 8080 but in order to run our code now you can see here we're dealing with running our client application these are two disparate entities on the on the web now so we need to be able to run both of these projects both our client and serverside code simultaneously so I'm going to create another terminal window to run the server side code so let's go into server and then magic stream movie server like this oops got that wrong cd server it's magic stream movies that's why that went wrong movies server like that okay let's clear the screen and we can run our code let's go go run dot like that so let's see what happens here when we go to our client code and run it so let's run so npm rundev like this to run our client code okay so we're missing something here okay so when we try and run our React code we're getting this issue here failed to resolve import B bootstrap disc CSS bootstrap and we we installed React Bootstrap but we didn't install Bootstrap we have to install Bootstrap in order for this to work so this will mean that we can include the you know the react bootstrap allows us to include various elements within our code various React Bootstrap elements like button and things like that but in order to use the actual Bootstrap classes and for this importation code to work we actually need to install Bootstrap separately from React Bootstrap and that's why we're getting issues here so I'm just going to press Ctrl C and to install Bootstrap we just type npm install bootstrap and that should resolve our issue excellent let's clear the screen let's try run our code again npm run dev okay looks good and let's launch our client code okay so that hasn't given us the same error but I think there is another problem and I'll explain what that problem is now we should if I'm not mistaken get a cause error oh okay so let's run our code mpm rundev our client code here we've still got our server side code running here at this point 8080 and let's copy this URL for our client React code and let's paste that here press enter and nothing's happening and let's see if there's any issues and there's no issues apparently so what could be going on okay so I strongly suspect that I'm not calling this fetch movies function we still have to call this function we've created it but we're not calling it anywhere so that would be the problem there so all we do to fix the problem is below the function so within the use effect hook we call fetch movies like this and so now when the home component loads this fetch movies function should be called so let's error fetching movies great so that's better okay okay so let's look and see what errors have been printed to the console and it's an Axio error and here's what I wanted to see so this is actually an expected error what's happening here is we're getting blocked by cause a cause policy which is by default active within your browser and so our client on port 5173 is being blocked on the server here so that we we're not able to access this movies endpoint by default first of all what is cause it stands for cross origin resource sharing this is a default security precaution built into browsers that serves to prevent malicious websites from accessing sensitive data from another domain and you see we're running the React client at a completely different disparit location to where we're running the server so we're running the client on port 5173 on our local machines and we're running the server side code on port 8080 so this is considered two disparit domains and then cause kicks in and stops our client from accessing the endpoints on our server so cause stands for cross origin resource sharing this is a default precaution built into browsers that serves to prevent malicious websites from accessing sensitive data from another domain so basically the client and server code are running on different domains so our browser is stepping in and not letting our client communicate with the servers not letting us call the endpoints on the server from the client so to resolve that we need to include appropriate code on the server uh so I'm just going to stop the client from running by pressing Ctrl C cancel out of that and we can stop the server from running too so let's go to this terminal and do the same press Ctrl + C clear the screen here and then let's go to our main method in the go code so we're returning to our go code our server side code and we need to write code to prevent cause and we can do that by explicitly including code to prevent cause within our serverside code within the main function the entry point function of our serverside code so firstly we need to include a package to deal with this cause related issue so let's use the goget command to install the appropriate cause package so that we can leverage the cause functionality to prevent cause from interfering with our client communicating with our server okay so to install the relevant cause package to help us prevent cause from interfering with our client communicating with the server we need to install this package so we can use the goget command for that so go get and then github.com sljin dash contrib like this and then slashcores press the enter key okay it's downloading the cause package for us excellent so that seems to have been successful and then we can include the relevant importation code at the top of our main file here so within our main package here we include this package importation code to import the relevant cause package so that we can handle cause appropriately okay okay so now we can create a variable called config and we can configure our cause related code appropriately cause config like this and then let's configure cause appropriately so configow I'm going to say allow all origins and I'm going to set that to true now note that we will have to change this code before we deploy it because allow all origins will not work if we're using HTTPS so to secure our full stack application we are going to use HTTPS when we deploy this code to the production environment so we won't be able to use allow all origins equals to true like this boolean field like that but for now let's include that for testing purposes we can we can do that for now okay allow all origins and then config dot allow methods so we're allowing the following HTTP methods with this line of code here string and let's include get post and we're actually just using get post and hatch within our code so I'm just going to include these methods here um this should be within curly brackets and not round brackets like that okay and then config dot allow headers like this equals to and it's another string array and let's include origin content dash type and authorization like that allow these headers okay and then config dot expose headers equals to string array so these have to be curly braces on content hyphen length like that okay okay and then let's configure router use and let's configure our cause settings dot new and then config like this okay this is a bit of a pain to have to set all this up to bypass cause but okay so one more uh configuration settings so config dot whoops dot max age equals to 12 times time do hour like that okay and you must make sure that you've got time package the time package imported here so that it can be used down here appropriately and the other thing that I haven't done quite correctly is these this code here must be above where we're setting our roots here so this code must be called before the root configuration code that we created earlier so that cause will be handled effectively okay great so now let's go to our server side let's go to the terminal prompt for to run our server code so go run dot excellent so it's listening on port 8080 so let's run our client react code and to do that we go npm rundev like that okay and we can now copy this URL to our browsers to test our code excellent so as you can see we've got round we've got around the cause issue and our movies are being displayed in kind of a gallery a grid within from within our React code excellent so that is a great start for the creation of our React client code okay so we've now created the functionality for displaying all the movies available on our Magic Stream website to users which are and these movies are publicly accessible to everyone but in order to stream them you must log on and authenticate before streaming the relevant movies so the movies are being displayed on the home component we're drilling down to the movies component here and then each movie is displayed each movie is being displayed here within a loop through the movie component here and this is the movie component here which is basically just a card which establishes the layout for each of the relevant movies and we've got Bootstrap here for styling the relevant cards displayed in like a card deck if you like or like a gallery with the poster images displayed in a gallery for each movie before we create the front end user registration and login functionality we need to add links to the front end so the user can navigate between the main pages we're going to establish the relevant links within our header component so the header will reside at the top of the website and will contain all the navigation to the main pages login registration home etc we are going to leverage the react routter DOM package to manage our client roots because the linking to the relevant pages is all um part of the rooting system so each page has to be configured within our rooting system and what's governing that rooting system if you like is react rout so this is a package we need to install react routed DOM so to do that let's go into the client forward slash client magic stream client like this and then within this directory we can install react rout and to do that we type npm install react router DOM like this and press the enter key excellent so we've now installed React router DOM and we can make use of that within our header component which we are about to create so let's do that let's create the header component so within the components folder we simply create a folder called header let's make sure we stick to protocol and the folder has a lowercase H at the beginning of the word header like this and then let's not make the same mistake as we made earlier with the movie and movies component and let's make sure that the JSX file first character in the name is in uppercase so we call this header with H and uppercase JSX like this jsx and we're going to create our component within this JSX file so let's start at the top and let's import the button from React router sorry React Bootstrap so React Bootstrap slash button so we can make use of the button element imported from React Bootstrap if you want to learn more about React Bootstrap please navigate to their website within your browser the React Bootstrap website where you can get more details about the relevant uh components available to you through React Bootstrap okay let's import container so we're importing the container element from React Bootstrap react Bootstrap and then forward slash container like this okay whoa I haven't yet mastered the art of typing and talking at the same time i am improving but uh just bear with me okay so then import Whoops see look at me import nav like this from And we're going to import the nav element like this react dash bootstrap nav and you'll see these elements will be placed in their relevant context as we develop the code for this component so okay so navbar and we want navbar import nav bar so all of our navigation is going to reside like for example buttons that link to the register page or the login page are going to reside within the navbar situated at the top of our front end okay so react bootstrap slashnavbar like this okay and let's now import the relevant components from react router DOM and this is these components are extremely important because this is how we can establish a centralized routting system for our gosh for our front end navigate so use navigate hook we want to use navigate hook nav link like this and link and you'll see how we use these in our code in just a bit from and react router dash dom like that brilliant okay let's create the infrastructure for our component so const header equals to we won't pass any props in at the moment but a prop for logging out of the application will be passed in at a later stage when we get to that functionality a little bit later so const navigate like this equals to and then I'll use navigate hook like that okay and then let's return I think whoops that doesn't look quite right something's not quite right it's because I haven't made that an arrow function and that's what all the fuss was about there and let's go return okay and then within round brackets we can include our JSX code so let's start with a navbar which we've imported from React Bootstrap here so let's include a navbar element like this and just bear with me while I include certain attributes here and I'm not going to explain all the attributes but you can as I said go to the Bootstrap 5 or Boot or React Bootstrap websites and you can read about and you can glean whatever details you wish from the content published on on those websites okay LG okay like this uh this should let's just close that nav bar what's the enter key so expand equals large okay then class name equals to shadow SM like that okay and then within the navbar let's include the container element i'm just going to create the tags and go navbarbrand but ordinarily Visual Studio helps us out and creates the actual tags for us when we press the tab key after entering the name of the element okay and for now for the brand I actually am going to include a brand i've created a an icon that we can use for Magic Stream brand but for now I'm just going to include Magic Stream here like this this is the name of our fictitious company so for now we'll just include that for our branding we'll get back to the the actual branding code in a bit okay let's include our tags first this is one with a dot in it also so bar dot toggle and the significance of the navbar okay and this is a we can close the navbar we can close the navbar toggle element like this okay get area controls equals to main navbar nav like this and these are just Bootstrap 5 classes that we're using here and the significance of the nav toggle is that on smaller screens the menu you can close and open the the menu bar using a particular toggle which is useful for smaller screens so on a desktop computer for example the browser being in full screen uh you won't need this toggle but it will appear automatically on smaller screens so that the menu bar can be opened and closed um as required so it's for responsiveness and that's basically one of the advantages to using Bootstrap in general is for screen responsiveness so that's why we're using this nav toggle this navbar toggle element here and let's create the actual collapse functionality so this is what we want to include in the collapse of the menu the items the navigation bar items that we want to include when the menu bar is expanded and collapsed so when the menu bar is expanded we want the following items included within this so if you want to create the tags and the element at the same time within VS Code can make your coding a little bit more efficient just type nav that's the type of the element and press tab like that and it'll create the opening and closing tags for you okay and then let's include class name like this equals to me auto okay so within this nav element this parent element we want to include our our link our home link so let's create our tags then nav.link like this and let's close that off nav.link so we're including a link within the nav element nav.link element we go as equals we want this to be part of the React router DOM functionality so we include the nav link element here and this is what we've imported here from React Route DOM and this is how we're establishing this as a link so Bootstrap knows that this is a link from React Route to DOM and will manage this linking functionality for us okay and then we include the to attribute and this is where we want this link to link to so the page will be just forward slash which is the homepage and then within here we're just going to include the label for the link which is just home okay so this is the homepage for slash the root of our navigation so when the user clicks on the home link it of course takes the user to the homepage okay excellent this is when the user logs on and the user wants to see the user's recommended movies and we'll create the functionality for this component a little bit later so recommended is the name of the page that we're going to create name of the root if you like that we'll create in a little bit later and let's label this as recommended like that okay brilliant um and I'm just going to temporarily create a state variable here so we haven't yet imported use state so we need to do that so let's import use state like this from React okay and I'm just going to create a state variable so for now I'm just going to create a boolean state variable for this purpose just for testing purposes equals to use state and I'm going to set this to false ie the user is not yet logged on okay and this is just for testing okay so we got those links home recommended which will be displayed to the user okay regardless of whether the user is logged on or not okay and let's now create the links within the navbar for the registration and login buttons and the logout button but as I say depending on what mode you're in but depending on whether the user has been authenticated different buttons will show up for the different scenarios and let's look at the details of this now so let's click let's type a nav like that so we've created our nav elements within this nav element we want class name equals to the following classes ms auto space align whoops align items center like this okay great and then let's include empty tag so a fragment here let's include a span so this is for the scenario where the user's logged in so I'm just going to type span close span and we're not doing the login functionality just yet so for testing purposes I'm just going to go hello and then within strong whoops within strong HTML elements I'm going to say name like this so hello name but this will be replaced with the username of the actual logged in user later on okay and then below this we're going to include the log out button so button so this is when the user is actually logged in let's go variant equals to and this is just for styling purposes we're including these bootstrap related classes out line like this and then size equals to SM okay and then we're going to include an on click but we'll include that later because we're not going to include the authorization functionality till a bit later so we'll leave out the on click function for the buttons for now okay and let's go log out there great and then so for this one if orth is truth the if our state variable is truth the then and then open brack open round brackets like that and we need to actually put this bracket at the end here like that okay so if orth and then question mark we want this true when the author is set to true or is truly we want these elements to be displayed okay and then here we close the round bracket and we include a colon and we want to include now what happens when author's falsy meaning the user is not logged on okay we need to remove this round bracket here oops put another question mark okay like that and that looks much better now we need to include the code here and this code will be for displaying the register and login buttons in fact I'm just going to copy and paste these in to save time and you can see here now we've got the register button here and we're navigating to to the login and we're navigating to the register form here sorry the register page here okay let's just tab that in a bit okay so if that means the user's been authorized we're displaying the log out button and if the user hasn't been authorized we're displaying the login and registr and registration button here and we you can see we've got an onclick event included within this button here so we need to we're going to in order to navigate to these pages which we haven't yet created we need to use the use navigate hook so we've got our navigate variable here which has been retrieved from the use navigate hook here and we're using that method here to navigate to the relevant pages okay we've got one little issue here this collapse element should be closed down here we want all of these as part of a collapsible navigation bar so we must include collapse element here we must close it down here okay and the other thing is the container element also close the container element down here everything all of these navigation links or buttons must be included within the container like this as well as the collapsible menu bar okay great nav should probably be tabbed in a bit okay okay let's make our code look beautiful okay there we go i think that's pretty good so if we go to app.jsx JSX let's include the header component here just for testing purposes for now so that we can see what's going on header from header so we're importing that oops and I don't think I've exported it yet so that's something that we can't forget so let's export header as default from this component header component default header like that and now we're importing it within the main parent component the app component here and then we can just include it above include the header element above the home element and we can see kind of what's going on now so for testing purposes this should be good um and let's open another terminal window here go CD server we want to run the server magic stream movies server hopefully that's correct it is and let's run the code go run dot we're running our server side code now and let's run our client side code so npm rundev like that okay let's copy this URL to our browsers let's see what we got here oops okay we got a problem failed to resolve import components oh home so let's leave that up there um okay right so we've not quite got that correct of course this needs to be header let's save that and see if the error goes away so far okay new dependencies okay let's just see what's going on here so nothing's displayed we have issues here see go to dev tools error caught use may be used only in React router DOMJS routter component okay so we can't actually use that yet okay let's resolve this right okay I understand what's going on there okay so we need to go to the main.jsx file here and we actually in order for those for that routting functionality that we've included in the header component we need to wrap our app component within a parent component that's taken from the React rout package so let's do that okay so let's import the following from react rout so import curly braces browser router we need to include that there and then roots like this and then root we need to include this code here so we need to take that out there include browser routter like this within the browser routter parent component we include a component or element called roots like this and then within the roots element we include a an element called root like that and we go okay and we can close this like that so let's remove that and then path equals to okay so for slashstar like that and then element equals to app like that okay and I think that is correct now um see what happens there we go look at that okay so we've got our login we got our register recommended we've got all of that now in place excellent we go to a small screen you can see what I was talking about with the toggle you can now toggle those open and closed if you on a smaller screen like that and we can neaten up all this a little bit later but yeah so our routing is pretty much working i mean our header functionality is pretty much in place that looks pretty good okay good okay so the next thing we want to do is complete our roots so to do that we're going to import the necessary components from React roottodom so root sorry that should be capitalized brute roots and then use nav gate from react rout okay and now we can configure our roots so this pretty much stays where it is so let's include the roots element okay and then our fish root and this is for the home component so path equals to so this is a route to the home page home component element equals to then within curly braces we can include our home element like this okay okay we need to include them within this tag like that need to include the root like that the home element within the for the element attribute set to the element attribute within the root element like this and this is how we configure our roots and react rout okay great let's create one for register 2 so path equals to register like this we're going to create a component for register in a bit like that there and let's do the same for login for the login route let's just make a duplicate of this route and change the relevant bits so login like that login um this should be capitalized login so we need to create these components so let's do that so within the components let's create a folder for register so red whoops this should be in lower case register like that and then within the register folder let's create a file called register.jsx register.jsx like this enter okay i'm just going to create just the basic structure for the register component we're not going to include any of the functionality just yet we just want to get it navigating correctly to the register and login pages first off and then we'll create the functionality for these two pages so for now let's just include very basic functionality so let's return just H2 uh register like that so it's just very basic functionality to include an equals here of course so we need to export the register component export default register like that okay and now we need to create the login component so let's go to components there so let's create the folder for the login component so login then within the login folder let's create the login.jsx file with the L capitalized login.jsx enter and I'm just going to go to register here i'm going to select all copy and paste let's paste that within our login.jsx file and change the relevant bits login export login login like that okay excellent i think that's pretty good so we've got the components in place so now we need to import them within our app.js file and create the relevant and configure their roots so header let's go red just like that reister range okay let's duplicate that and do the same for the login element login component we're importing the login component here like that excellent and now we can create Oh we've created the roots of course so they're already in place and we should be able to navigate to those roots now we should have very very basic rooting functionality in place now and we certainly don't need that anymore the counts that was added by default through vit we should be able to test that now let's just see what's going on there so our serverside code is still running excellent so let's run our client code mpn npm rundev like that and let's copy the URL paste it here okay look at that looks pretty good and let's see if we can navigate to the login page login excellent so that routting functionality is basically working let's go to register login let's go home so we got home we got login we got register and now we just need to create the functionality for the login and register components we've created our header in its most basic form and now we want to create the login and register functionality excellent okay so in order for users to make use of the magic stream service a user must sign up with the application to do this the user will access the page that we are about to create and provide the user's relevant registration details first name last name email address password as well as the user must also choose the user's favorite movie genres thriller drama comedy action crime that sort of thing our system needs this information the genre information in order to recommend movies that the user may find most appealing that is one of the core services that the Magic Stream website offers a recommendation service so once the user registers with the user's relevant details the user will be able to log in to the application by entering the user's email address and chosen password the user will then be able to make use of the applications services okay so before we create the registration form we're actually going to go back to the server side code and we're going to create an end point where we can get all the genres and if you look at Compass quickly here I'm just going to launch Compass let's connect so if you look at the Magic Stream Movies database here and we go to this collection called genres you can see that I've created a collection that we imported earlier on in the course this collection contains all the relevant genres that will display within a multi select box within a multi selection box that the user so the user can select multiple genres before the user registers with the system and as I say this is all part of the registration functionality the user will select one or more genres that the user most appreciates so the user basically wants movies in the relevant chosen genres recommended from our recommendation system that's what it basically means so firstly we're going to create a endpoint on the server before we actually create the client side registration functionality we're going to create this this server side endpoint within the movies controller here movie_controller.go so the the um the controllers package within the controllers package within the movie_Controller.go file we're going to scroll right to the end here and we're going to create a handler function for an endp point that can be used to retrieve all of the genres that are stored within this genres collection here okay great so let's do that let's create a function and this is just a basic function nothing too hectic let's go funk and let's get genres this is the name of the handler function and we want this to hook into genonic so jin dot handler funk like this we've done this all before let's open curly braces and let's return an anonymous function that accepts a jinggonic context as an argument so a C parameter of type jin.context context is part of the function definition of the anonymous function that we are returning from the get genres handler function and we're going to include the logic within this anonymous function here so this allows us to hook into JgonX web framework and map this function to an endp point that the user can access from the client via HTTP okay so firstly we do the usual let's just in fact I'm just going to copy and paste this code and I'm not going to offer any explanation but we know that this is part this is a way for the resources to be managed so once the relevant queries for the MongoDB for go driver have been executed there may be resources hanging about in memory and this ensures that whether the query fails or works that the relevant resources are cleared out of memory appropriately so this is just housekeeping code that we need to include here and let's create a variable called jean and it's of type model so it's an array of model dot genre types and it's in fact models so that should be models do.re and this is where we're going to store the movie genres that we retrieve from the genres collection that resides within our magic stream movies database okay excellent and of course if we go to the models package so we go to movie.mmodel we've got the genres the definition for the genre type or strct here excellent so let's go back to movie_controller.go and let's complete the logic for our get genres handler function okay so we need to create a cursor as we've done before and we want to create an error object here and then let's use the operator and genre collection and I'm not sure we've created a genre collection yet find Okay let's just check if we've got a genre collection here i don't think we do we haven't got a genre collection defined so let's do that let's duplicate the rankings collection here or ranking collection and let's make this genre genre collection and let's call this genres because that is the name of the collection within the database here genres so we're creating the collection variable here and we can go back down to the bottom now and continue with our code genre collection dotfind and now we've got IntelliSense great and then let's pass the context here and then bson.m so we want an empty bson.mm here because this is to do with the bson collection that we are retrieving from the database this is the type it's an bison format that we're collecting that we're retrieving from the genres collection within our MongoDB database so that's what that is there and then we do the usual go thing we check if error is not equal to null meaning we need to handle an exception caused by executing this query we can handle it like this c.json and open braces htt http.status internal server error comma and then jin.h we're using uh we're using generic jin functionality here to return an error in JSON format to the client and let's give it an appropriate let's uh include an appropriate error message error fetch fetching movie movie genres like that okay brilliant and then let's include the return keyword here so that no further code executes from within this get genres method and then this is all part of the resource clearing code defer cursor so all the resources related to this cursor must be cleared and this is what we're doing here let's pass in the context ctx we're making sure that those resources are cleared no matter what happens within this function whether an error occurs or it's successful we need at an appropriate time we need the resources associated with this cursor cleared from memory and that's just good programming practice and let's carry on so if equals to and then cursor now what we're doing here cursor we're calling the all method and we're going to put we're going to put what's in the cursor into the genres array that's basically what we're doing here and remember the amperand refers points to an actual memory address where that data will reside so the genres the genres variable will point to a memory location where the genre's data will reside okay and then we continue on the same line here so we're checking we're we're returning an error object here it'll be null if this is successful or it will contain a value if it's not successful and we can actually using this semicolon we can check the error on the same line like this so if it's not equal to nil here open curly braces and we can handle that exception http dot and it's just a status internal server error and let's use jin to return wellformed JSON to the database uh to the client okay and we'll just return what'sever in the error whatever gets returned in this case from the error method here which is called on the error object okay this should be cursor and let's press the enter key and include the return keyword there and then we're in good shape if the code gets to this point and we can return JSON with a HTTP status okay http sorry HTTP dot status okay and then the genres collection and we've now written our end point and let's now go to the unprotected roots where the unprotected roots have been configured um so we go to the roots folder here unprotected roots and let's configure it and it's just a get request so it's get and this will be just genres so this will be the path part of the path to the end point so it's obviously the endpoint consists of a base path which will be localhost colon 8080 when we're testing it on our local machines forward slash genres to get those genres and then we map it to get genres the function the handler function that we've just created okay that hooks into Jenonic excellent so now we can test that so I'm going to run the server code let's go to this terminal window here and type go run dot to run our code okay so it's listing on port 8080 and let's go to Postman and just test that endpoint is valid is working correctly before we create the registration functionality so here let's replace this with genres like that let's change this into a get request and let's run it okay and has it returned the genres it has look at that we're in good shape excellent so let's move on and create our registration client side registration functionality so we can press Ctrl C to cancel the Genonic web server listening on port 8080 and let's go back to the client code and now we can develop the registration functionality so let's do that let's go to the client code here let's go into src components and remember in the last section of the course we created this registration folder and the register.jsx file and within this file we're going to now create the registration form and you'll see a little bit later how we need to call the genres endpoint to retrieve those genres and then we're going to display those genres in a multis select box that the user can exploit to select the various genres before registering with our system but let's take care of the basics first i'm actually going to do a lot of copying and pasting i've prepared this code offline we don't necessarily need to go through all of this code line by line it's just going to take too long to do that so I'm going to just paste in all of these importation lines of code here so we're importing state use effect container button form we're going to be uh creating a registration form so we're going to make use of this react bootstrap form element here and then of course use navigate and link we'll make use of the use navigate hook and the link element and then let's go through this fairly quickly so I'm going to just paste in this code here so you can see here we are creating a state variable for each of the fields that the user must fill in when completing the registration form here so first name last name email password confirm password the genres in the set genres function here will be to do with the endpoint that we've just created the favorite genres and set favorite genres are to do with the genres that the user selects from the multi select box that we'll include on the registration form okay great so I think we've got the gist of what we're doing here let's move on so let's just remove this JSX code here create a container here and within the container we'll include the form so let's close container off here okay so let's let's include the form and I'll just explain what's going on here i'm going to just include all the fields within the form within our code okay so I'm going to copy and paste that in there so this is the form that denotes our registration front end form here so we've got to handle submit we have to handle submit once all the fields have been filled in by the user we've got the first name field here and we've got a placeholder within the text box enter first name we've got value equals first name here so this is the state variable that we've created up there and then we've got set first name as the as to to set that first name state variable here when that text box changes and the pattern is exactly the same for the last name and then same for the email and we've got a form control and you see this code is all available to you in the relevant GitHub repository and that GitHub repository URL is included in the description below and you can just literally copy and paste this code um so I don't want to go through and explain every all this code line by line because I think it's fairly self-explanatory and you can always look up the various elements from the React Bootstrap website or the other Bootstrap websites that exist where you can learn about Bootstrap and styling and layout and that sort of thing and CSS of course that's not what this course is really about this course is more about developing a full stack application using Jin Gonic and React so it's not necessarily about the styling and um all of the front-end code that goes with it that's more ancillary to the end goal of this course so please by all means look up all of this code here if you're not sure of what it's doing okay so here we go we got favorite genres.m map so this one needs a bit of explanation form select here this is where we're creating that genres that collection of genres within a select box that the user can use to select the genres that the user prefers over all the other genres and that's to do with the recommendation service that is offered by Magic Stream so genres.m we still need to write the functionality to call the genres endpoint so we can populate this genres collection here on the client and then it's adding the various genres to this select box here and you can see each option denotes a genre excellent and then down here we've got a button to actually register while the users information is being processed on the server spinner is displayed within the actual register button here and that's what that code is there and uh the the button will be disabled also while the the various fields are being processed but as I say please go to the go to where the code is hosted on GitHub and you can just copy and paste the code I'm not going to go through and explain this line by line because that will just take too long and then we've got onsubmit equals to handle sububmit and we need to write that method but Before we do that let's include a header and I'm just going to paste it just above the form here okay brilliant we actually need to close the div that div is associated with that div there and we need to close the div here just above where it says container here and that's just a header okay brilliant excellent so create your Magic Stream Movie account is the header there um we've got a logo image which we will include later so I'm actually going to just delete that for now and we'll create that a little bit later so that we can brand our website appropriately okay but let's get in the basic functionality let's get the basic functionality working for now okay and then above return here now we want to include our call first of all to the get genres um function and we're going to use a use effect for that so I'm just going to paste in this code here you can see here we're including the use effect hook we're calling the genre's end point and we're populating the genre state variable here and of course if an error happens we're just handling that error by writing the relevant error to the console window and then of course we have to call the fetch genres method that we created here from within the use effect hook and what this means here this is an empty array so when this component the registration component or the register component first loads we want this to immediately get called and populate the genre state variable there okay so we created that use effect there so we need to create a function here which is being called um handle genre change so every time a user selects makes a different selection within the genres selection box we need to update the favorite genres variable and this is what that is doing here in this code handle genre change so we're updating the favorite genres state variable which stores an array of genres or a collection of genres and it's set whenever that selection changes within the select box on the registration form the set favorite genres is appropriately being called here from within this function and it is changing the various options within the or the various collection items within the favorite genres collection that's what this code is doing here and we're calling that you can see here form select it's a multiple select box and we're calling that every time the value changes so on change it sets it to handle genre change and that's the function we've created here and it updates the favorite genres array and that function is handling that and then we need to include one more function it's very important function and this function handles the event when the submit button is clicked by the user okay here so you can see we've got our various settings here it's creating a payload of the users's details so it's creating a variable called payload first name last name email password and then the ro default role if a user is using the registration form the default role is user an admin account can only be created on the server so if it's created through the client interface through the website then the default role is always user so that's what that code is doing there oops just gone a bit far there so user we're setting the default role there and the payload is being set appropriately the favorite genres are being set to the favorite genres here we're creating the payload and we're posting that to the register endpoint all of these all of this data here we're posting that to the register endpoint and it's being processed and then we're navigating to the login page here okay excellent so we've got our registration code pretty much set up might be a few little things we need to adjust here within the handle submit um form here like navigate to login i don't think we need to do that right now oh we can do yeah we can just navigate to login okay once the user is registered and I think we can potentially just test that now let's see what that looks like anyway I think we've got everything okay okay and let's let's run the server code first let's go to the server terminal window oops and let's run let's firstly clear the screen and let's go go run oops go run dot to run our code okay let's go to the client code and let's go npm rundev like this and V is now running our client side React code on our local machines and we can now we should be able to now test our code we've got the server side code running and we can go through our browsers to the client side website should load up our movies great so we've got our movies loaded here now okay and let's see if we can register a user let's go to register look at that so that's looking pretty good and you can see here we've got our genres here and we've got instructions on how to use this genres select box here okay that doesn't look quite right there genres oh it is okay so hold control in brackets Windows or command in brackets Mac to select multiple genres so if we hold down the control key if we're on a Windows machine you can see how we can select our genre so let's select comedy western fantasy and thriller why not okay so this user likes all these ones here and I'm going to register i don't know have I registered Gavin Alon yet i don't know if I have let's go to users bob Jones Sarah Smith Ben Madison Craig Denton oh I have okay so Gavin LS let's select someone else let's select Goth you can always tell when a name is made up can't you goth Jenkins goth Jenkins just sounds like a madeup name okay anyway so Goth Oops goth i bet you there is a GT Jenkins out there though jenkins at Hotmail whoops let us know in the comments if your name is Genkins genkins atotmail.com okay genkins password okay i'm just going to select password one exclamation point confirm password one exclamation point so we just stick to one password just makes it easier to test our code and then we've selected our genres here and let's see if we can register this okay so register okay so hopefully that's worked it's registered the relevant uh user Goth Jenkins we can we'll check that out in Compass in just a bit and then it's navigated us to the login screen which we haven't yet developed and let's see if Goth Jenkins exists now within our users collection okay so let's go view reload data goth Jenkins look at that and you can see his password has been encoded appropriately he's part of the user group no tokens yet because he hasn't logged in and we've got his favorite genres here comedy western fantasy and thriller awesome and now we want to create the login screen so that GTH can log into our system okay so let's do that that's the next step i'm going to cancel this here ctrl C to cancel when you've when uh it's listening on your local machine just C through Visual Studio Code cancels that operation and now we can go back and now we can develop our login screen so let's go to login here and I'm going to do the same thing i'm I'm not going to explain this line by line because it just takes too long and this course is already huge so okay so anyway let's paste in these this importation code oops this importation code here so we got use state container we got all these react bootstrap react elements that we're importing here we got we're importing axios client and the various react routed DOM elements and this use navigate hook here okay and let's let's develop our login code let's do a similar thing as we've done before so we're going to include our state variables here okay we've got a state variable for email password okay right and we got a state variable for email and password okay there's not so much to fill out in this form we're obviously just logging on using a unique email address that we chose when registering and the password which at the moment is just password one exclamation point with the P capitalized we're just sticking to a standard password okay and then we've got all these various things here we haven't written this use thing that's part of the next part of the course so I'm removing that okay we'll keep all the rest here and the form looks something like this i'm going to copy and paste the form which I've already obviously developed and you can just paste this from you can copy and paste this from the relevant GitHub repository the URL to that GitHub repository has been included within the description below pasting in that code there okay and all it is really okay we don't want this yet because we haven't yet branded our website so I'm just going to delete that for now welcome back please log to your account that's the header right there and then we've got these groups these React Bootstrap groups for each of the fields email address here so value equals email state variable when it changes when the email address form control changes we have to set that email state variable accordingly and this happens before the user hits the submit button to log in to the magic stream website we've got password it's the same pattern here excellent and then we've got this button here for submitting to the login endpoint submitting the payload information which should be the email and the password to the login endpoint here and we've got a link that links the user to the register form if the user has not yet created an account using the register form okay so that's all that is happening there okay we go along we're not yet using these we're not yet using this use location hook or this use navigate hook so let's create the actual code that interfaces with the login endpoint so I'm just going to copy that from the code I've got off screen and paste it in here and and this is the handle submit function that interfaces using Axio's client with the login endpoint we're passing in an object a JSON object with the user's unique email address and password so that the user can be authenticated with the system and then it sets this or object here um okay and I'm going to create a temporary orth object here um up here this is going to be replaced in a in an another section of this course in a later section of this course but for now just for testing purposes let's set the orth object with the with some user information that gets returned from the server once the user's logged in and we'll do that now so we'll set that or object accordingly so set response.data that's what's returned from this post request that we are calling here we're calling the po the HTTP post method here and interfacing with the login endpoint via HTTP and passing this object which contains the email address and password here and then we're handling an exception if one occurs here um and here we're if all has gone well we're actually populating local storage but before that we're setting a state variable here and this will be covered in in a subsequent section of this course but for now I think this is fairly good and I don't think we need to use this navigate variable or we can basically just once the user logs in um actually what I want to do is just write that to the screen i'll console log that to the screen oh we're already doing it here so we'll be able to see what's returned from the Axios client so I think we're pretty much ready to test the login functionality now okay excellent so I know I'm going through this really quickly but a lot of this is pretty self-explanatory you can see this is of type submit so in the form here in the form elements here we've got an onsubmit event which is wired up to the handler function here handle submit and here's handle submit and we've been through what this code does here it interfaces with the login endpoint and passes in the payload which includes the email address of the user and the password to authenticate the user with the system let's actually test that let's go okay we need to go back to the server we're at the server here let's type go run dot to run our code so it's listening on the server and let's go to let's Whoops let's actually clear the screen npm rundev like that okay let's copy this URL to the browser okay great and our movies are loaded up we've registered Genkins let's log in as GT Jenkins look at that that's looking good okay so go jenkins at hotmail.com and the password I selected was pass word one exclamation point and let's see if we can log in okay so at the moment it's not doing much but we can test whether we've logged in by looking at the console window and seeing what's been returned okay let's go to our developer tools and it's returned an object and it's returned a token and a refresh token that is working perfectly excellent so we're able to log the user in but we will handle you know we can we'll handle the the once the user authenticates bit so once the users authenticate we'll handle that in a better way um so it'll go for example in the header that's the next thing to do is in the header it'll go welcome goth for example or hello goth or whatever this login and registration button will no longer appear here one log out button will appear so the user wants to log out once he's logged in at some point so it'll have a log out button instead of the login and register button there and of course if the user hasn't been authenticated the login and register button are displayed there this functionality this mechanism is working it's returned uh the appropriate information back to the client once GT Jenkins has authenticated and it's returned a token and a refresh token which can be used for subsequent requests to the server to protected endpoints so now that the user has been authenticated the user will have access for example to the recommendation the movie recommendation service and will be able to stream movies also because the user has been authenticated so when the server side code is when the protected endpoints are subsequently called this token will be part of that request and the user will be able to be authenticated on the server which means that this user now that the user has been authenticated the client will be able to call protected endpoint so the recommendation service for example can be consumed by the user that's just been authenticated okay excellent right so we've created the login functionality so now it's time to adjust the front end appropriately once the user has been appropriately authenticated the user's logged on it's authenticated and we want for example the header to change so we want the two buttons that would be displayed when the user hasn't logged on the register registration button and the login button we want those to be displayed if a user who's accessing the magic stream website has not yet authenticated and when the user authenticates i.e logs in we want a logout button to be displayed and not the login and registration button to be displayed so we're going to handle things like that now so what's the effect on the front end when the user logs into the to our magic stream website so if you'll recall let's go to the login page here we're creating this or We're setting the orth object here once the user has authenticated we just did this as a temporary fix we want this orth object to be available to all of the components because we want the component as it were to know whether the user has authenticated or not at any given time so this isn't ideal this is just localized to the login function we want this orth to be available to all of the components and one way we can do that is by creating a React context okay context is a way to share data between components without having to pass props down manually at every level of the component tree a problem often called prop drilling so here's an example of where we're actually using prop drilling so you can see here we've got the home component we're fetching all the movies and then we're drilling down these movies to the child component so firstly the movies component we're sending the movies object down to the uh we're passing it down as a prop to the movies component and within the movies element within the movies component we're looping through all of the movies that have been retrieved from the server and then we are calling the movie element here and we're passing down the movie prop so we're drilling down again to the movie element or movie component here and then we're displaying the movie so that's kind of like an example of prop drilling whereas with context you'd just be able to retrieve the relevant objects from a centralized location so it's kind of like it's a data store essentially in memory that you can just retrieve those relevant objects at any given time within a component you can retrieve it from a context and that's a much more efficient way of doing it especially as your application grows you want to use context rather than prop drilling to pass the relevant objects between components okay so we're going to start by creating an orth provider and this is how we are going to create our context whereby we can pass the orth object between our components without needing to use prop drilling for this purpose so let's create a folder let's start by creating a folder within the src so it's going to be at the same directory level at the same level in the uh folder tree as components and so within src we're going to create a folder called context and let's name this with a small C context text like that press the enter key then within the context folder we're going to create a file called orth provider and this is how we are going to create our context functionality or provider.jsx Okay and let's import the relevant components from the React package so we want the create context function imported from the React component and the use state hook from React like this whoops okay and then let's go const let's create a variable called orth context this is our context and then we create our context by calling this create context function here so create context and then we open round braces and pass in an empty object which is signified by opened and closed curly braces like that okay and then let's export so we're creating the actual context component export const proider this is the name of the component equals to and then within round braces we pass in a prop called children and this represents all the components in our application that will be wrapped and you'll see how we do this a little bit later within the orth provider element so think of this children prop as all of our components so this will enable all of our components to retrieve the relevant orth object from the children that's basically what this it's very abstract at the moment but this is how you need to think of it for now to understand it and let's open curly braces like that const and let's include within square braces or comm set orth and this is the the orth is a state variable of course that's going to be passed between our components use state let's use use state the use state react hook like this and we've got the set or function here of course which can change the orth state variable that's what we can use the set all function for then let's include the return keyword like this open round braces and then within tags here let's create the tags for the orth provider dot provider element like that and let's close it properly here so we're closing the orthro provider.provider provider element there and then within within we need to include a value attribute here and this is how we are essentially passing down the relevant or making available the orth object and the set function to all of the children within our application so to finish the job we include children here this children prop here within curly brackets within the parent orth provider.provider element let's save that and then we export default like this or context so we can utilize the or context context great okay and we save that and then we want to create a hook so that we can easily access this context this O object and set or function from within our component so to make it easy we're going to create a hook a react hook a custom react hook and hook will be let's create a folder called hook at the same level as context and components so within src let's create a folder called hook like this with the small h like that and within hook let's create a file called use orth and you'll see how we can use this use hook in just a bit jsx okay let's create the hook it's pretty simple also let's import the use context hook from React and let's import the or context from the provider that we've just created so this has the path to it context forward slashorth oops or provider like that and then const let's create the hook const use or equals to open and close round brackets and then arrow function and then we return the use context hook and we pass in the orth context like this this is how we can make available the orth object and the set or function to to the components of our application let's export that export default use or like that semicolon this is all to do with making the orth and set the orth object and set orth function available to the components within our application so we need to wrap all of this code with the orth provider so these are going to become child elements within the orth provider and that's how our components can be placed into their context where they'll be able to use the use or hook to retrieve the orth object and the set or function okay so within the strict mode element here let's include or provider but we need to import or provider first so import open curly braces or provider and then it's automatically detected the path from where I want to import or provider excellent okay and let's move this space neaten up our code a bit save that crl S and then we can wrap these elements the roots and everything within the O provider element so the or provider element becomes the parent element here and this is how we can pass down the relevant context that contains the orth object and set function to the child elements the child components of our application okay the next thing let's go to the login component here within components let's go to within login here open login there and we need to retrieve the orth and set the orth object and set method from our use hook so first thing we want to do is import the use or hook here so to do that let's type import use or like this from let's include the appropriate path which is this hooks for slash use like that brilliant and then we can use the use off hook down here we need to comment this out we no longer want this here or in fact we could instead of doing it like this we could simply you just use the set or because we actually only want the set or function we don't need the orth object here because we want to just set the orth object so we're going to go set off within curly braces like this and retrieve that from our use or hook use like this and then we don't need to pass in these this empty string so remove that and we've got the set or function available to us within our login component and we're using it here set to set the orth object but now you see we're setting it globally not just locally so by using the hook here we're hooking in to the context we've abstracted a lot of the complexities of how the context works and we're hooking into the functionality using the use or hook here and then we're using the set or function here okay brilliant so we got that sorted out okay and for now I'm just going to use the navigate to navigate to the homepage just for now we'll change it to something more appropriate in a bit but so we just navigate once the user is authenticated we'll navigate the user to the homepage where the movies will be displayed just for now um but later on what we actually want to do is if they go to if the user isn't authenticated and for example wants to utilize the recommendation service so clicks the recommendation um menu option on the header if the user is not authenticated immediately what we want is for our front end to display the login form so that the user can authenticate and then once the user authenticates we want the user to be navigated to the recommendation page the recommended movies page rather than back say to the homepage so that's just for a better user experience but we'll put in that functionality a little bit later for now let's just once the user logs in let's navigate the user to the homepage okay that looks good to me and let's go to the header now and sort out the header functionality okay so the first thing we want to do here is import the use or hook import use or Okay and it's detected the correct path to the use or hook here so we've got the correct importation code in place now and then we want to include again we need to transform this here we're going to use the orth global object here by using the use hook use or hook so we need to change this so within curly braces let's remove that or oops or from and then use or and then we don't want to pass false in there anymore this is not a boolean variable this is an object containing the user's information which is retrieved from the server once the user authenticates so it'll include for example the user's token that will be passed in with subsequent calls to protected roots so the user will have access to protected roots because the user has been authenticated so we've got the orth object here retrieved from use or here excellent and now this actually should just propagate through appropriately here hello name but we want within curly braces here we want to include the variable we want to include the value or dot first name so this is one of the fields that will be retrieved from the server be first name on the orth object when the user logs in and now we have access from the header component to the orth object which is now globally available through the context functionality that we've built okay excellent and then the rest of the code in fact should be good and we'll include the actual logout functionality a little bit later in terms of the layout we need to include a little bit more functionality okay so within the components we're not going to include this within its own folder we're just going to include a file called layouts layout.jsx like this so it's got a capital L there to stick with convention naming conventions layout JSX so let's import the layout element import sorry let's import the outlet element from within react routed DOM so we can hook our layout into our centralized routting system which is made possible by the react routed DOM package here so const layout let's create the layout component and you'll see how we'll use the layout component in just a bit but let's just create the component for now go return and then open round braces and then main element type main whoops main tab and then within the main element this is just a the representation of the main semantic HTML element and then within the main element we include this outlet element like this okay excellent oh I was wondering what all these red squiggies were about it's because I haven't included this as an arrow and the red squiggies go away that was just an equals whoops that was just an equals causing all these red squiggly lines and I've turned the equals into a arrow because this is of course a component denoted by this arrow function and this component is called layout and let's export the layout component so x Whoops export default and then layout like whoops layout like that okay and we need another layout related component and this one is going to be called uh require or and you'll see the significance of this as I write the code so required or dot j whoop jsx enter and let's write the code for the required or component let's take care of the importation code first import then within curly braces use location navigate and outlet like this from and then within quotations react router DOM okay and let's import our use or hook from and let's include the appropriate directory hook for slash use or like that and let's create the component const like this equals to open and close round brackets an arrow and then let's open and close curly braces where we can include our logic for our required orth component or let's retrieve or the or object from the global store if you like using the use or hook like this and then con location for navigation purposes from the use location hook which has been retrieved from the react router DOM package here so we can hook into that functionality for navigation purposes and then let's return so if or if all is truly then open round braces and we just want to include the outlet element here that we've retrieved from react rout that there okay let's close we've closed the let's close the round braces here for this true part here and then else and let's open round braces here and then we want to navigate so we're using the navigate element here we want to navigate the user to the login element so so if the user is not authenticated and tries to access a protected route we want to navigate that user to the login screen so that the user can authenticate let's include the state attribute here and the from location so from location and then replace here so it will replace whatever is in the whatever page is displaying within the layout it will replace that with the login okay remove that there okay we've got all these squiggly lines here oops navigate let's just figure this out so all Okay we need to close off this navigate element here okay and that all looks good now so basically what this is saying is if the user is not authenticated the user for example tries to access a protected endpoint if the user is not authenticated we navigate that user to the login screen so that the user can enter the user's credentials and authenticate with the system okay okay and let's export default and then the required required orth component like that okay now we want to sort out our roots that's the next step and hopefully this will start to make sense to you what we're doing here so let's go to the app.jsx file here we're importing all of these components at the moment here but we need to sort out this code here path equals to forward slash homepage and element now equals to component that we just created called layout so we pass in that layout element okay so those are our nonprotected roots now we want to include our protected roots within the required orth element that we just created so this is how we're protecting those roots so you can see now the significance of required all so any of the roots included within the required or element if the user is not authenticated will be navigated to the login page where the user must authenticate and that's it significance here so let's go back to the app.jsx JSX file here and let's create a parent route for the protected roots element equals to and then within tags here required or like that and let's close off that route here and let's include the route here for recommended within the protected routes okay let's include path equals to recommended like this and then element equals to and then this is a component that we're going to create we haven't yet created it rec re receded like that and this is just to give you an idea of how we are going to protect these roots so if the user is not authenticated the user automatically gets presented with the login screen and then the user must authenticate in order to access the protected route here that's how that works i'm going to remove that for now because we haven't yet created that component and I want to just test what happens with the header when we now authenticate so I'm just going to remove that there okay excellent um and let's actually run our code let's see what happens when we run our code so let's run our code let's navigate to the server like that cd magic stream movies server like this okay and let's go run sorry let's go go go run run dot to run our server side code let's run the client side code npm rundev okay I'm just going to copy that URL to the browser okay great so we've written quite a lot of code and some of it's quite complex so this is what I would have expected we've got a few issues to take care of so let's look at the first one failed to resolve import it's not finding that path okay so let's I'm just going to stop this i'm going to exit the browser there and stop running the client as you can see we've also got our error defined here so it's telling us where it is it's within the login component here okay and I can already see what the what the issue actually is here um anyway I'm just going to cancel out of that sorl C clear the screen okay and we need to just fix this and it's because I've actually named this this should have been called hooks because you want to you know potentially include multiple hooks within the hooks folder but I called it hook so I'm not going to rename it here because we've had issues in the past when we've tried to rename folders and files i'm actually going to keep this as is even though it's not ideal it should perhaps be hooks rather than hook so I'm going to just rename this here this path to get around this issue right let's run the client again dev let's copy that let's copy that URL to the browser window press enter we've still got problems let's go to our dev tools to see if anything's been logged that can be helpful to us okay render orth provider there's a problem with orth provider let's go to the relevant i'm just going to close that down um and let's go i'm going to cancel out of this actually and I'm going to go to orth provider and what could be wrong here ah so this is incorrect orthrovider.provider this should be or context so I'm using the wrong element here so it's orth context here okay let's save the change let's run our code again mpm rundev our serverside code is already running so we don't have to run that let's copy that

URL to the browser window okay let's go to our dev tools layout is not defined okay I think I know what that is it's just because we're not importing the layout element from within the app.jsx file within the app component so let's close that off and let's stop our code from running again c clear the screen and let's do that so let's go to app.jsx and let's import the layout element with this importation code import layout like this from then within single quotes let's include the relevant path so component forward slash layout like that okay let's see if we've got any more issues mpm rundev Okay let's copy this URL URL our browser so that we can run the client and the server side code is already running so we don't have to run the serverside code and we still have issues let's go to our developer tools and see what the next issue is oh required or so we're not importing that within the app.jsx file so let's go out of that and let's control C this clear the screen and let's go to app.jsx JSX and then we just need to import required orth the required orth element here which uh navigates the user to the login page if the user is not authenticated and tries to access a protected route okay so let's go to app.jsx here we've duplicated the layout importation code here and let's change the relevant parts so that required orth is also being imported correctly here or okay and that looks good let's try run our code again dev mpm rundev and let's copy the URL to our clipboards let's go to the browser and run that ah okay so that's all working now and I want to quickly test the effect of logging in on our front end i'm hoping that these login buttons will disappear once the user's authenticated and the logout button will be replaced in the header here or I mean will be displayed within the header here the logout button rather than login and register so let's log in and I'm going to log in as Genkins atotmail.com this is the user that we recently registered genkins password one exclamation point let's log in excellent that looks good that's exactly what I wanted to see okay so it's clear that we're now logged in and we can log out using this button but we haven't written the functionality for that we will write that functionality in just a bit okay brilliant the next thing I want to do is create the functionality for the recommended component because this is actually a protected route and then we can see how our application behaves when accessing a protected route when the user is not authenticated and then when the user is authenticated okay so let's go out of this here and let's write the code for that so let's Ctrl C this clear the screen and within the components folder let's create a recommended folder like this and within the recommended folder let's create a file called recommended.jsx JSX to house our recommended component okay great so let's create the relevant code and actually before we write the logic for this recommended component we need to create another Axios file another Axios configuration but this time where we're configuring the header with the correct authorization part to it which will contain the the the token the user's token once the user has been authenticated the user is given a token that the user can then the token can be included in subsequent HTTP requests so that the user can be authenticated against protected endpoints so the user can access those protected endpoints if the user has the correct privileges so we're going to create to accomplish this we're going to create a new file a new Axio configuration so let's do that we're going to create an actual hook so that it's easy to retrieve this Axios this private Axios configuration where we're passing in the token so within the hook folder let's create a new file and let's call this file use cate whoops sx like this and let's write the code for this Axios object so that we can configure this Axio object so that it includes the relevant token the authorization token within the header of subsequent HTTP requests once the user has been authenticated with the system so firstly let's import Axios from Axios we've installed Axios so we can write the code like this so let's write a hook called use Axios private use aios private equals to an arrow function and then let's include the relevant code okay so let's create const axios or equals to axios dotcreate like this and we're going to create our config code here okay and then of course we want the base URL so we're going to read in the base URL from our environment variable so in the environment variable we've got the base URL configured there um so let's first read that in so we go to do that we go const API url equals to import meta nv this is how you read in environment variable values um when your react project has been created using vit api_base url like that whoops url like that okay and that should read in the appropriate environment variable value for the base URL from here let's go back to our hook that we're creating our hook code so now we can configure that base URL by typing base URL colon then API URL like that it's the base of the API URL and now what we're going to do is create something called an interceptor whenever this Axios object attempts to call an endpoint it's going to add certain information to the header and in this particular case the information we want is the token the O token the access token so that the user can authenticate against protected endpoints okay so to do that firstly let's retrieve the orth object and set function from our use or hook and we aren't importing it just yet but we'll do that in a bit use like this okay let's import that use orth hook with this code here import use orth okay and it's found use in this path that's excellent automatically it's done that okay and then what we need to do is create an interceptor as I said so with this interceptor with this interceptor it's going to intercept a call to a HTTP endpoint and add and we want to add relevant header information so that the user can authenticate against that protected endpoint before accessing that endpoint so let's go axios or dot in to seps response this is how we add an interceptor to an axios object dot use and then open round brackets then we want to pass in a config variable like this config round braces and then arrow function okay we want to configure the header we don't need that round bracket there we want to configure the header so config.headers headers do authorization like this equals to within quotations bearer space curly braces we can include the variable so we need to precede these curly braces with a dollar so we can include a variable within the single quotes okay so now we can use the or object and dot token like that h and sorry these need to be back to characters because we're including the variable within the curly braces so let's remove those single quotes and include back to characters instead of those single quotes here like that okay great okay so we only want this configured if of course the author is true these so we need to include an if statement there so if or like that and then open curly braces like that we want this code to reside within there if the orth is true the if the orth object is truly the like that and then we return config because we've configured it appropriately for axios like that excellent that looks good now and then outside of this here for the axios for the actual hook we want to return axios or so we include the return keyword at cos like that semicolon and that looks good okay and then let's export this as default export this hook as default so default use axios private like that so it means the user must be authenticated to use this access component and then the or token can be included within sub within HTTP requests once the user has been authenticated so instead of using this we want to use the this Axios object for protected roots which will include the token within the header of the calls to the relevant protected endpoints okay great that's all that means there and then let's create our recommended component which is of course a protected route so we're going to want to use this hook to get the relevant Axio object to retrieve the relevant Axio objects let's go back to recommended and let's write the code so firstly let's import use axios private from dot dot slash dot dot slashhook so it's just hook not hooks forward slash use private oops for slash use private like that so we've got the correct importation code there let's import use effect and use state from react like that let's import the movies component from this location from movies slash oops forward slash movies like that great and let's create our component const using our arrow function recommended it's the name of our component then arrow function and within our arrow function we include the logic for our for our component so first thing let's create a movies state variable and set movies function to change the state of that variable equals to use state that pass in an empty array so an empty array for the movie state variable we're defaulting to that firstly and then let's create a loading variable loading and set loading function equals to use state and we want to initialize the loading state variable to false firstly and then const message then set message equals to use state like this okay and then we want to retrieve our AIOS object from the hook that we just created so Axios private equals to use Axios private so it's use Axios private hook like that and let's create a use effect let's use the use effect hook like this so when the component first loads we want to call the recommended movies endpoint so that we can retrieve the recommended movies for the relevant user and it's from a protected endpoint okay let's create the function that fetches the movies we're going to call this fetch reccommened movies okay fetch recommended movies equals to a sync open and close brackets arrow function like that oops error function like that oops set loading to true like that set message to empty an empty string and then let's include a try catch block here okay and then const response all right let's just include the catch section here catch error and then let's handle the error by just console dot error method and we'll just write that to the console screen so error fetching recall mended movies like that and then more detail about the error and the error object we can include that in what's written to the console screen when an error occurs okay and then finally so this code gets called whether an error occurs or not and we'll set loading to false so we don't want any loading indicators displayed once this process has resolved whether an error has occurred or not we want to set the loading we want the loading indicator to disappear at that point okay great so then response equal oops equals to await axios private.get and then for slashremened movies that's the name of our end point there and then we need to set movies set the movies state variable to response dot data whatever is returned bear in mind we're using the Axios private Axios object here and we've created an interceptor so when we call a protected endpoint the token the orth token is being sent along with that message that HTTP request to that protected endpoint so we're so the user is being authenticated against that protected endpoint the recommended movies endpoint i just want to double check that that is the name that we called our endpoint so we can go to roots protected roots and recommended movies is the appropriate end point and we've tested all of these endpoints through Postman so we should be able to test that fairly efficiently test our client fairly efficiently at this point because we know the server side code works okay there's that our use effect hook written we need to call the use effect sorry we need to call the fetch recommended movies function within the use effect hook so we do that like this okay and we need to pass an empty array and this is to do with how the life cycle hooks work in React so we want as this when this recommended component loads we want it to fetch the users's recommended movies okay great okay and the return jsx is pretty simple that's the most complex we've just written the most complex part of this particular component and so we're checking the loading state variable loading so if loading is truthy then while our component is loading let's just include a very basic loading indicator we'll include a proper spinner loading indicator a little bit later but for now let's just include text like this loading dot dot dot okay else then open round braces we include the movies element we pass in the movies state variable we're prop drilling here I guess you could say we're passing the movies and if there's an error message we're also passing the message down to the movies component here and we've got loads of red lines here we want to include this we did we don't have a parent element so let's make this as a fragment which allows us to the ability to not have to include an apparent HTML element like a div for example so we can just uh get round that by including a fragment here which are just which are denoted by empty tags and that looks pretty good and we mustn't forget to export our component it's easily done export default recommended so now we're exporting the recommended component as default which means we can import it here within our app.jsx file so let's do that so I'm just going to duplicate that there let's import recommended from components recommended and then with the R capitalized recommended like that okay and now we can include within the protected roots which is wrapped by this required or um element that we created as the parent element for our protected roots and we can create a dedicated route for that recommended element so recommended is the client path to the recommended component so the client route is defined by for slashrecall mended and then that's a path to the recommended component there and it's wrapped within the required or element because it's a protected route so in other words if the user is not authenticated and they try to access this route the login screen must be presented to the user indicating that the user needs to authenticate with the system before accessing the recommended the relevant recommended movies okay and that looks pretty good i just want to double check that within the header it is indeed that link is indeed linking to this path here so if we go to header here let's have a look at what's going on recommended nav link yep recommended and that's what I want to see so we can now test that i think it should work fairly well although actually I'm going to go to the login screen the login component here and include this code for the navigation so it doesn't navigate the user when the user authenticates say the user's on the homepage the movies are displayed the user wants to check the user's recommended movies the user's not yet authenticated the user presses the recommended header menu item the user is then navigated to the login screen when the user logs in we don't want to navigate the user to the homepage again to see the movies we want the user to go directly to the recommended screen so let's comment that out and this code here will do that it should direct the user red yeah it should log the user on and then direct the user to the screen that the user was trying to access before the user was authenticated so that just makes for a much better user experience by doing it that way and I think there's a piece of there's more code that we need to write to make that work not sure I've got that in there yeah okay so we need to also this from variable here we need to retrieve that appropriately from use by using this location variable and I'll show you how to do that now so and that's retrieving the relevant uh path that the user was trying to access before the user authenticated so let's do that const from equals to location dot state dot from dotpath name like that and if it's null whoops and if that is null so there's no we can use we can navigate the user to the homepage so if this is actually null this path name which is taken from the browser's history if this is actually null it will by default navigate the user once the user is authenticated to the homepage okay excellent so I think we're in good shape there and I think we're ready to test it we've still got our server running okay great so we don't need to rerun the server let's run the client mpm rundev okay great it's running hopefully there won't be too many issues with this could be some issues okay let's go to the client okay that's looking good so far so now we're not authenticated and let's access the recommended menu option here okay so the first part of it the functionality is working we're not authenticated so our code has navigated us to the login screen and let's let's now authenticate as go jenkins@hotmail this is hotmail address his email address.com so go jenkins@hotmail.com password one so we are including Genkins's credentials within the signin form and let's try and log in and let's see what happens okay so we got a problem and that's okay and let's just investigate this by going to the developer tools so more tools developer tools okay so it's responding with a 401 exception and I suspect that's something to do with the interceptor not adding the token into the header before making the call to the protected endpoint the recommended movies endpoint so it's a problem with Axios so we kind of got an idea of where to look so let's close that down let's go to our code here i'm going to controll C that cancel and clear the screen and let's take a look at use Axios private here and I can already see what the problem is here i included response here and this should in fact be request we're making a request so every time we make a request to an endpoint a protected endpoint we want this our orth token to be added appropriately to the header so this should fix that problem and let's see what happens now if we run our code so our server side code is still running so npm rundev let's copy this URL to the browser let's launch the our browser window run our code brilliant so our movies are being displayed to us but this is a nonprotected endpoint for the homepage it's just the movies endpoint that is being called to retrieve these movies and then displaying it on the homepage which is correct and then we want to see our recommended movies but of course we haven't authenticated with the system so the system doesn't know who to recommend those movies to and we haven't of course authenticated whether we have the privileges to use the recommended movies facility so we need to authenticate so go genkins oops at hotmail.com password one exclamation point and let's log in and let's see what happens excellent so that means our token has been passed to the server after we've authenticated and the system has recommended the movies based they're all excellent movies apparently the sentiment attached to all of these movies so it's recommended the best movies in the genres that we chose which were western comedy thriller drama I believe th those sort of genres were selected when and we registered as Goth Jenkins so it's recommended the appropriate movies but this is big because we've now from the client we're now able to access not just protected endpoints not just sorry not just unprotected endpoints like the movies endpoint which displays all the movies on the client for us in the on the homepage but we've also now passing in a token with requests to protected endpoints like the recommended movies protected endpoint so when we press recommended the recommended when we select the recommended header menu option here it automatically authenticates us on the server because the token is being passed in with the request to the relevant protected endpoint after we've authenticated with the system the token is passed down to the client the client then stores that in memory and then when we subsequently call protected endpoints the token is used to authenticate us so we can now just because we're authenticated we can now go swap between protected endpoints we can call protected endpoints and unprotected endpoints so we can now call unprotected endpoints and protected endpoints no problem once we have authenticated because that token can now be passed in with subsequent HTTP requests to protected endpoints and we can authenticate through the use of that token that access token excellent okay so in this part of the course we are now going to create the UI part associated with the AI functionality so this is when the administrator logs into the system and will have a facility available to that administrator and the administrator can then create a movie review for a particular movie and then when the user submits when the admin user submits that movie review that movie review is sent along with a base prompt to open AI it is anal an analyzed and through the prompt the instructions are to extract one word describing the sentiment behind the movie review so this is a way for us to extract quantitative information from qualitative information the qualitative information is the movie review written in natural language and that is becomes part of a prompt to the AI the AI analyzes that and returns a just one word and then we're associating a a value with that word so uh the options the word options to describe the sentiment that we have included in the prompt are excellent good okay bad and terrible and we you can see that it's very easy for us to associate a value with each of those words so excellent um has an associated value of one and terrible has an associated value of five okay is value associated value of three bad has an associated value of four and two is the value that represents good so it's a way for us to rank the movies in a sense so through this ranking you can order the movies by those values and that's exactly what we're doing for the recommendation system so we're recommending five of the top movies within a users's genre using those rankings to order the movies those five movies and if for example there's more than five movies available for the users's chosen genre it will only recommend the top movies based on those sentiments those rankings extracted from the admin's movie review so let's get started and in fact I'm going to keep this very simple we're going to get this done really fast because I'm not going to write out all of the code because most of it on the front end is just Bootstrap uh classes and CSS basically just to style a UI facility um uh the code is available on GitHub so please go to the GitHub repository if you want to copy and paste the code and then you can do your own analysis on the various Bootstrap options by all means please look them up on the various Bootstrap websites so that you can uh get more detail about those classes and what they're doing but it would just take too long for me to explain every single Bootstrap class and uh I think it's unnecessary for this particular course so as I say I'm just going to copy the and paste the code for this particular UI facility and then I'll give you a brief explanation of what everything is doing so let's um create within the components folder let's create a new folder and let's call this review with the R in lowerase so review like this and then within the review folder let's create a file called review.jsx and I've already written this code it's on GitHub i'm actually going to copy and paste the code into this file and here is the code just pasted it in here and I'll explain the code to you now okay so this is what the front end is doing we're using the Bootstrap the relevant Bootstrap classes to basically split the screen in half on the left hand side we've got the actual movie which will it'll display the poster and we're reusing the movie element for this purpose and this is just the various Bootstrap classes that are used to we're using the Bootstrap grid system to place the movie on one half of the panel the movie poster and relevant details including the sentiment and you'll see how that works when we run our code uh we can change the sentiment on the fly in real time by updating a text area box which appears on the right hand side so that takes up the other half of the screen the right hand side of the screen is taken up by this text area box so it's just a big text box multi-line text box that where the administrator can add the administrator's review like this so it's a text area type for this form control we're using React Bootstrap for the form control element here and then we've got the submit review button here which is a submit button type equals to submit and we can submit the form value the form control value for the text area which contains the administrators movie review to the server to our review endpoint or what have we called the endpoint the end point is update review and we're sending that along as with a patch request to the server and the relevant IMDb number of the movie so that the server knows which movie to update with the relevant admin review so of course we're using Axio's private because it's a protected endpoint remember in the last part of the course we're behind the scenes we're adding in the users token in this case it would be the administrator's token that the that is received from the server side when the relevant user logs in so the token is returned to the client and then we are including within this functionality here use Axios private this hook we're including an interceptor and we're adding that token to the header so that requests from that particular user can be authenticated on the server through the use of this token which contains the user's claims which is just information about the user and it includes the user's ID for example the user's role and that sort of thing so the user can be authenticated on the server and therefore access protected endpoints and we've taken care of that behind the scenes here within the configuration of this particular Axios object that we're using here within the review.jsx JSX file to send a patch request to the server to the update review endpoint where the administrator is updating the administrator's movie review and then what's returned is the ranking name here which is the sentiment behind the movie that AI is extracting so please by all means refresh your memory and go back to the serverside AI functionality that we created which sends this movie review along with a prompt to open AAI um and the open AI functionality extracts a sentiment from the movie review basically so it analyzes the movie review written in natural language and extracts one particular word and we can attach a numeric value to that word so the words are excellent good okay bad and terrible for but they can be extended you can create an entire spectrum of rankings so this uh software is designed to be extensible in that regard so you could add as many uh ranking values as you like or sentiments sentiment options as you like um for this functionality so that's basically it really and so if we just to describe what's happening here we are analyzing the orth state variable here this is the or uh variable that gets populated when the user authenticates but we also have to check the orth ro that it's admin in order for that particular the particular admin the particular administrator to have access to this text area control so the user can update the administrator's review um or else what happens we've got an else part here we're just putting that review in readon mode so any user can it doesn't have to be an administrator to actually see the review but the user has to be an administrator in order to update the review so here we're just so here in this particular scenario the code is deemed that the user is not an administrator and is displaying the relevant review within a an alert control and this is just a bootstrap class which turns this div into an alert so it's basically read only you can see the review in in readonly mode but if you're an administrator you can actually update that review and send it through to the server to be processed great okay and we haven't yet created the spinner code here um I think it might be a good time to do that actually and I'm going to do the same this code's not that important it's just for um a better you user experience i'm actually just going to create a new folder called spinner here and I'm going to create spinner.js okay and I'm going to copy in the relevant spinner code and it's just basically a span and a div and utilizing various CSS properties to create the spinner effect okay and we are within the review we are importing spinner up here and then while the relevant UI screen this component is loading that spinner will display so while this loading state variable is truthy the spinner displays and once the data is ready the loading indicator is is disappeared and the actual UI is displayed to the user where the user if the user is ad if the user is an administrator can update the relevant movie review okay okay and now we need to update the app.jsx file here so let's go to app.tjsx and it's in a protected endpoint so we need to firstly import the endpoint so I'm just going to copy this here duplicate this code and then change the bits the relevant bits so So this one's called review okay review review so we're importing the review here okay review is declared but its value is never read so we're going to do that now we're going to utilize this review component within we're going to add a root for that particular component so I'm going to duplicate this here path equals 2 and then it's going to be review like this and we need to actually include a parameter within this path and it's the IMDb parameter imdb ID parameter because when this route is accessed we need to pass an a unique movie ID so that the movie can be extracted the relevant movie data is extracted uh from the MongoDB database and the relevant data is then displayed for the movie on the review screen okay that's how that works and of course we need to replace this recommended with re view okay we've mapped that route now that looks good okay excellent and then we also need to include a review button for our movie cards but we don't want that review button we're actually displaying a movie card we're reusing our movie element on the administrators panel where the administrator updates the relevant movie review but we don't want the review button displayed in that regard and this is just uh we're going to use prop drilling to to accomplish that so let's do that um update review okay this is worrying me there's Okay already included file name Golang magic stream view okay I've done it again i've done something stupid sorry okay I don't know why I do that that's got a small letter in it i'm going to just cut this here okay and I'm going to delete this file here and I'm going to recreate that file sorry about this it's got to have a capital letter here view.jsx let's stick to protocol and make sure that that letter is capitalized can't believe I've made that mistake again okay sorry about that and let's paste that code back in there and that should sort out our problem on the app.jsx JSX page great that has Sorry about that we were importing sorry here it is we're importing the review here and there's no red squiggly lines there now okay great so and we've got our root setup and the next thing is to we've pasted in the update movie review method here which simply navigates to the review page and passes in the relevant IMDb ID value to this route here and then that parameter can be utilized within the review page so if we go back here you'll see how that parameter Here we go so we're using the use params hook to get the IMDb ID value here and we're importing it from react routin this Axios private get query we're getting the relevant data for a particular movie which will could potentially include the review for that movie and the ranking the sentiment and all the movie information for that movie the title the movie poster etc and we're grabbing that from this movie endpoint which must include as a parameter the IMDb ID value of the relevant movie so this is how we're passing that to the server the relevant IMDb movie then it queries our MongoDB movies collection and returns the movie data and then we're setting the movie data here the state variable called movie we're setting with the set movie function here great and then we're reusing our movie element here we're reusing that element here and displaying the relevant movies information in this particular UI okay let's go back to the prop drilling code um I got a little bit distracted by that error um issue we had on app.jsx so what we want to do is drill down this value here this method we want to drill that into the uh home element first okay and this is because we only want the movie button the review button displayed in certain context and I'll I'll describe those contexts in more detail in just a bit but let's just drill this down using prop drilling let's drill it down to the home component and we're sending our method down there so it can be used well it can be used within the movie component ultimately but uh through a button that the button when it gets clicked will call this navigation functionality through this method call so we're passing that down to the home component so we need to include this prop within the home component let's go to the home component here okay uh we include it here we include the prop here okay we don't need those curly braces okay there and then we actually want to drill further down to the movies component here so we're going to do the same thing here we're going to include the prop the relevant prop here paste that in there so we're passing down that method reference the reference to the update movie review method that resides actually within the code for it for which resides within the app.jsx file or the app component here but we're drilling it down we're passing a reference to that method down firstly to the home component then to the movies component so this is some serious prop drilling so we need to include it here okay and then we can drill down further into the movie component so we do that like this equals to like that and then we must include this prop within the movie component here okay and we can do that like this and now we're passing in that update movie review method and now what we do within the movie component is create an if statement where if update movie review is truthy include the button and in this respect we can control when the button is actually displayed for the relevant movies we can control that by checking whether it's truthy or not so if it's passed down as a prop to the movie component then we include the button if this isn't included as a prop the button won't display so we can control when that review button is displayed and then we're calling the update movie review method here and passing in the relevant unique identifier when that button is clicked this code is executed here on the app.jsx file here and we're using this navigate utility here to navigate to the review element and we're passing in the relevant IMDb ID like that so the relevant movie review is displayed the relevant movie information and the movie review is displayed on the review page and I think we're in good shape i think that's all the code we need for now let's let's run the code and see where we are with this okay so I'm going to run the server side code and we do that by typing go run dot like this press the enter key excellent okay okay so the server side code is running let's run the client npm rundev press the enter key to run the client code hopefully we don't have any issues here but I guess there's always a chance that we do anyway let's copy that to our browser window so that we can test out our code okay we have a problem okay failed to resolve import hooks ah that should be hook okay no problem it's because we copied and pasted the code in from the prototype type code that I wrote in preparation for this course that's what's causing that so uh where do I go here i go to review so let's go to review and let's change this to hook there's another reference to it down here okay save that mpm rundev there any more references I don't know about let's run that okay let's copy that to the browser window and let's see what we got okay excellent you can see now our review our review button is being displayed under each of these movies so that has worked perfectly and now we should be able to click the review button to review a movie okay actual review code is not working for some reason just check what's going on here navigate is not defined i'm just going to clear that um let's go to app.jsx here we're not uh retrieving a variable we're not retrieving a value from the use navigate hook yet so to do that we can type const navigate equals to use navigate like that i don't know what I was thinking i' i've included this outside of the function to include that within the app function of course okay mpm rundev okay let's copy that to our browser window let's try again review great so it's prompting us to authenticate let's just log in as goth jenkins at hotmail.com password one exclamation point let's log in and there we go i love the acting in this movie it was absolutely sublime and why isn't the text box showing in this particular case and the submit button well that's because this user is not an administrator it's not part of the admin role but part of the user role so all the relevant movie information is being displayed and the sentiment excellent but we can't change that sentiment through updating this movie review because we're not an administrator so let's log out oh we don't have the log out functionality yet written do we okay so we can't log out um let's go home let's just cancel out of the whole thing and we'll write we'll have to write the log out functionality soon let's clear the screen and let's run the client again but this time we are going to log in as an administrator and let's check who's an administrator i believe it's Bob Jones he's always an administrator so let's go into the compass utility here and check our data to see who we have designated as an administrator i believe Bob Jones is an administrator let's check and then we can log in as Bob Jones he's an administrator bob Jones okay great and you can really make any one of these an administrator because you have access to the back end just change the role just double click in there and you could change for example Sarah Smith to an administrator i won't do that now but you can easily do that if you want to test other users as administrators but let's log in as Bob Jones because that's an easy thing to remember all Bob's credentials are pretty easy um we're still running the code so let's go to our browsers and copy that in there and let's see if we can log in as Bob Jones now and let's go to Once Upon a Time in Hollywood and let's review Once Upon a Time in Hollywood let's click review let's log in as Bob Jones bob Jones at hotmail.com and then password one exclamation point login and look at that this movie is awful and it's not awful at all it's a Tarantino movie and it's really quite a good movie i really enjoyed it but we've decided through this admin review that the movie is awful so now let's change that let's uh say um another masterpiece by I don't know how to spell terino i think that's right another masterpiece from Tarantino tarantino another masterpiece by Tarantino the acting was also sub blime sub sub blime let's see what it extracts from here as the sentiment so let's submit the review there we go we got our spinner working it's come back excellent so we've changed the sentiment to excellent from terrible to sentiment through our admin review and isn't that great you see let's let's try another review and this time let's do a rather middle of the road review so let's say um it wasn't great but it wasn't awful either let's see what sentiment we get from that okay middle of the road so that's working pretty well this movie wasn't too bad actually I'm hoping this will bring back the sentiment of good good so our AI is behaving itself our AI functionality and we're able to add reviews because we're an administrator we logged on as Bob Jones and we can add our reviews to our movies now and it's extracting the appropriate sentiment which has a bearing on our recommendation functionality here which brings back the recommended movies and the users genres the chosen users genres and it brings them back in order from one to five so all of these are excellent so it's ordered Highlander 2 is apparently excellent okay we've got to do something about that review cuz this was a horrible movie um so it's bringing back Highlander 2 as excellent Pet Detective excellent and then Once Upon a Time in Hollywood we've just changed that review as good and then Hangover is good and Harry Potter as okay so it's extracting the five top movies in that particular user's chosen favorite genres and that is working perfectly so it's ordered them by ranking which is the sentiment and we've associated a value from one excellent to terrible five and that means we can order by them quite easily here so that's our recommendation system is working really well there but let's do something about Highlander 2 i'm not accepting that that was excellent so let's go back let's go down to Highlander to excellent no it was an abomination and I'm going to I'm going to reflect that in my review so this movie is an absolute tempted to use a some colorful language here but I won't i'll just say this movie is an absolute abomination with an exclamation point so that should be terrible now let's submit that through and see if our AI deems this movie is terrible and it does good this is a terrible movie and you can see now the effect it'll have on the recommendation system see now Highlander is not even mentioned here in our recommended movies for Bob Jones because we've decided Bob Jones is an administrator and he's decided it's terrible so it's been unceremoniously kicked off this page completely great so it's no longer being recommended as a as a excellent movie as a movie and you can see that these movies have been ordered by the ranking value excellent good good okay okay brilliant excellent so we can't log out yet that's the next thing we need to do is build the login the logout functionality but I'm really happy with what we've done so far and I think that looks pretty good and it's pretty easy to use and it's responsive too you can see here if we make the screen smaller it just pops that under there like that which is pretty good okay excellent and we can just use our menu like this so if we go to recommended it just plunks those this is the beauty of using Bootstrap our screen is responsive our screens are responsive because we used Bootstrap and it handles that for us automatically so I'm really happy with that brilliant so at this point we have in place all the functionality or most of the functionality for our application but at the moment we are passing our access token to access protected endpoints from the client through the header of the HTTP message so if we look here for example if we look at use Axios private you can see here with this interceptor when we try to access protected endpoints we first are passing we are first adding the O token to the header of the relevant HTTP message every time we access protected endpoints so this is actually a vulnerability to sophisticated cyber attackers the reason for this is at this point we are currently storing the token in memory and in many cases when including the access token within the header of the relevant HTTP messages to access protected endpoints you may be tempted to store for example the access token in local storage on the client this is a security vulnerability because sophisticated attackers can potentially read the access token from local storage or even from memory using malicious JavaScript code so to overcome this security vulnerability we are going to store the access tokens using HTTPON cookies so why is it safer to handle the passing of the access tokens from client to server using HTTPON cookies well this is because HTTPon cookies cannot be read through JavaScript code and therefore this is much safer this is a much safer way to secure the access tokens so what are HTTPON cookies a HTTPON cookie is a cookie that cannot be read via JavaScript in the browser for example through document.cookie document.cookie cannot be used to read the cookie it's set automatically by the browser with every request to the server from the domain that set it the HTTP only flag makes it inaccessible to scripts and the secure flag ensures it's only sent over HTTPS why are access tokens sensitive access tokens are essentially keys to the user's account if someone steals them they can impersonate the user how can they be stolen what are the major threats xss cross-sight scripting this is where an attacker injects JavaScript and reads tokens from memory local storage or session storage csrf cross-sight request forgery this is where an attacker tricks the browser into making authenticated requests so how do HTTPON cookies help protection against XSS if you store an access token in local storage or session storage any XSS vulnerability can let an attacker read it directly with HTTP only cookies the token is not exposed to JavaScript so even if malicious scripts run they can't read it automatic sending with requests the browser automatically includes cookies with each request to the back end if same site rules allow so your front end doesn't have to manually attach the token to each API call let's create the code that will now store the access token as HTTP cookies so firstly we need to adjust uh what is occurring on the server side so let's go to user not user model user controller let's go to the login code here okay we want login user and there it is and you can see currently we are passing the tokens down here and we don't want to do that so the first thing I'm going to do is remove this this code so firstly I'm just going to copy uh comment these out we still want to generate the tokens in the same way we still want to update the database with the tokens so we're generating all the tokens here we're updating the tokens within the database and then we're passing the tokens back to the client this is the part of the code that we definitely want to change and how do we do that so we do that with this code here okay so just above where we are sending the HTTP response back to the relevant client let's set the tokens here so we are no longer passing that token back to the client like this we're passing all the other information to the client and we'll see why this is useful in a bit when we address the client side code in this respect so here we go we're setting the cookie access token here we're setting a timeout here okay and we've set the secure flag to true here http only true so it's a HTTP only cookie that we that we are now sending back to the client and this happens automatically so we don't have to manually send it back like this with this code here through the response it's going to automatically handle that for us and then we also want to do the same for the refresh token so we're setting the refresh token here to an appropriate HTTP cookie HTTP only cookie and the access token and we'll address what the significance of the refresh token is in just a bit we've done so we've now changed the server so that the access token will be passed from server to client via a HTTP only cookie there's a few things we need to do on the client to make this work so let's go to the client code we need to adjust the code here for the the first Axios object so it's actually a very basic update we need to include this with credentials flag and set it to true like this and that will ensure that the HTTP cookies are passed between client and server and we actually need to do it even though this particular Axios object is not used for protected endpoints we still need to ensure that the client and server that the tokens are being passed between client and server every time we send uh a message to uh the server from the client we need to ensure that those tokens are sent via HTTP only cookies um so we all we do is we set this with credentials property to true and we need to actually do the same now in our hook here where we've got the interceptor here we no longer need to intercept and pass in the bearer token like this but we are going to use an interceptor a little bit later for now I'm actually just going to copy this i'm going to comment this out i'm just going to ensure that with credentials is set to true here also like that okay so that is now correct so we'll leave this as it is for now and you might be asking yourself what's the point of even using two separate Axios objects but we'll address this in just a bit why I still want to maintain this separate Axios object for accessing protected endpoints and it's to do with the protect it's to do with the refresh tokens and we'll look at that logic in just a bit but for now let's just keep our logic as is here um in fact we don't need this line of code here either the with credentials property is set to true and that will ensure that the HTTP only cookies the relevant tok that store the relevant tokens are passed between client and server okay automatically so we don't have to add anything to the header and therefore potentially risk a security vulnerability to XSS attacks so we're no longer adding the bearer token here okay let's go to our login client now now that we've sorted that out let's go to our login client here and I want to make sure that we're storing it in local storage so that's what we want to do we want to store the orth data so this contains the orth data here you can see that we are setting the orth state variable which is now global we made that global we made that global through context um in the last part of the course so we're setting it here with response.data so after the login occurs we can see here on the server if we go back to our login code oops we can see on the server that we're passing this information back but we're no longer including the tokens we're just passing the user ID first name last name email RO and favorite genres back to the client from the server once the user logs in but this these cookies here these HTTP only cookies are passed back automatically so we don't have to manually do that through code because they're by virtue of the fact that they're HTTP only cookies and we're setting the appropriate flags on the client we're saying please do that on our behalf please just send those HTTP cookies automatically and we're setting with credentials equals to true here and we're doing the same for this Axios private hook that we created where we were using an interceptor to add the relevant token to the header when accessing protected endpoints we're no longer doing that we're just setting with credentials equals to true because we want those tokens to be passed between client and server automatically and this is how we do it but this is still relevant because we're going to um adjust the code here later on so that it handles refreshing the tokens but we'll look at that in just a bit we'll look at that later on so okay so let's go back to the login here and we're setting the orth object here we're setting the state variable well let's just first check that our code is still working we haven't messed anything up um okay so I'm going to create a new terminal here and let's go cd server slash magic stream movies server like that okay and let's run our code go run dot so we're running our server side code hopefully we have no issues good it's listing on port 8080 let's go to the client terminal and navigate to the relevant directory so client slash magic stream oops client like this okay and let's run our code so it's npm rundev to run our code okay and let's copy this now to our browser window and hopefully I haven't introduced any bugs error fetching movies oops first bug okay so interesting let's see what's going on there developer tools okay so we've got to address cause we've also got to change the cause policy um appropriately okay so let's do that let's handle that code let's go out of there let's cancel out of here crl + C clear and we got to handle cause okay so I've got to also cancel Ctrl C cancel the server from running okay and let's sort out cause we got to go to the main.go file to do that so it's this code this configuration code here we need to also change because we're we're um going cross domains with our client and server which gives you a lot of flexibility if it's on the same domain you basically have to always have your code on you know running on the same domain here we can run our components on disparate domains which is what we're doing here but the cost of that is we have to configure cores appropriately to make that work so let's do that and I'm actually just going to copy and paste the code for that this code is available on GitHub so please by all means just do the same thing just copy and paste the code from GitHub if you don't want to type it out manually just to save time so I'm going to copy this from my code off screen here and I'm going to paste it over here just above where we're setting up the roots okay great and okay we need to import log there we're using log and os of course because we're reading the environment variable but we need to configure the environment variable because what we're actually doing here is building up URLs that we want to permit to access the server basically so we're going to include our client side URL for the react code within the environment variable uh and we can we can store multiple allowable clients in that environment variable by using a comma delimited string that stores a delimited string of the allowable client URLs the the the URLs that we are allowing access to our serverside code so to do that we need to go to env and configure the relevant URLs so here are our allowed allowed origins here so we've got HTTP localhost 3000 HTTP localhost 5173 which is the one we're currently running our client on and then HTTP localhost 8080 so these ones are irrelevant for now but you could run your client from these addresses running on port 8080 and port 3000 and our cause configuration would permit these clients to access our server um so let's go back to main.go so we're reading all of this this information in we are creating this origins array here and we're just logging them so that we can test the code to make sure that it's working correctly and then we've got we're setting allow origins we can't use the other way we can't just allow all origins like we were doing before we have to allow specific origins because we're now using h the HTTP only cookies method and we want to when we publish our code to the cloud we want to use HTTPS so that our code is so that our data is encrypted on the wire between client and server that it's exchanged between client and server so this is another way to prevent our data from potentially being stolen so we're going to use HTTPS instead of HTTP so that our data is encrypted on the wire when exchanged between client and server and in order to do that and use HTTP only cookies we need to configure our cause in an appropriate way and this is what we're doing here as I say please the code's quite involved by all means just copy the code from uh GitHub and then go through it in your own time line by line so that you make sure that you understand exactly what's going on here okay brilliant so we've now configured course correctly let's run our server code again go run dot Okay excellent let's go to the client code let's clear the screen clear whoops clear npm rundev okay let's copy the URL let's just make sure that that URL we have indeed permitted that URL that we're running here 573 and we need to make sure that within our environment variable yeah 5173 so we are indeed permitting that origin to access our server side endpoints okay so let's copy that to the browser address and hopefully we won't get an error now when we try to access our movies endpoint and now that is working correctly but let's see if we can still authenticate the user a particular user so let's log in cray no it was go Jenkins let's use go genkins that was the last person that we registered go genkins at oops at hotmail.com password with the pc capitalized one exclamation point okay excellent so that's now working correctly we've authenticated goth oh okay oh so it is saying hello i was wondering what was happening there but the font isn't black so it's being hidden there so we'll have to sort out that problem hello go okay and then we also need to do the log out write the logout functionality let's go to recommended okay so we got a problem here cuz we're using the Axios private um object to access that endpoint and there's a problem so let's have a look at what's going on more tools developer tools 401 okay so something is going wrong there it's not allowing us to interesting okay so it's not allowing It's not authenticating it's giving a 401 unauthorized error it's not authenticating Genkins when he tries to access his recommended movies okay so the recommended movies endpoint so what could be happening there let's check let's investigate this let's I'm going to cancel out of this clear cancel out of the server code here clear okay okay i mean that looks good so this can't be a problem um must be to do ah okay h of course okay okay so what I haven't done is I haven't addressed the middleware so or middleware so it goes through this middleware code when it before it accesses any of the protected endpoints you can see here the protected roots routter use middleware middleware so it's running the code within here so let's go to definition and see we're going get access token let's go to get access token let's go to definition there and of course we're still reading it from the bearer token which is not what we want to do so we need to change this code get access token okay and it's pretty simple the code for that is pretty simple so let's comment out this code here i'm just going to keep that commented out so you can still have a record of how you would do it if you were passing the token through the header of the relevant HTTP message and I'm just going to paste this code in here token string here uh like this so we're reading the the access token from the cookie now instead of from the header that's what we needed to do there and that is the crux of our problem so let's see if that has solved our problem by updating the get access token utility function here so we're getting it from the HTTP only cookie instead of the header okay of the relevant HTTP message so let's run our server side code go run dot let's run our client side code i'm first going to clear the screen so we can see what's going on and then I'm going to run it so let npm rundev like this okay I'm going to copy this to the browser and hopefully we don't have any issues but you never know okay so getting our movies great so let's log in go genkins at hotmail.com password one exclamation point we're logged in let's see now if we can access the recommended protected endpoint excellent great so it was the way we were getting the the token within the middleware the middleware protects those protected endpoints so we were still trying to get it from the header and we're not we're not storing the token in the header anymore we're storing it in a HTTP cookie for security reasons that we've just discussed great so those are Goth's recommended movies so we're able to access endpoints and that is really good so the next step is to write the logout functionality let's do that now okay so I'm just going to cancel out of that ctrl C to cancel out of that when you're running the client and it's the same when you're running it through VS Code on the server just control C let's clear the screen and now let's write the logout code so we're going to actually write this function within the app component and we're going to drill it down using prop drilling into um the header I believe yeah we want the log out we want obviously the logout button resides within the header so when the button is clicked we want to run that logout functionality and we are going to store that logout functionality we're going to store the method that handles the logout functionality within the app component okay let's create the function for handling the logout we'll call it handle logout and I'm just going to paste it in here as I've stated a few times now you can just paste in the code from the code that's stored on GitHub and uh you can copy it from the relevant page on GitHub and just paste it into your app.jsx code here okay so I've pasted in handle log out here and we're going to pass that down through prop drilling to the header okay to the header component so and we can do that through this line of code but obviously we need to include the prop within the header component so let's go to the header component here okay we need to include that within curly braces handle log out okay and then we need to include the logic within the button so the logout button here we need to include an on click and we can do that very simply with this line of code so we include an on-click event and a non-click event handler that we're passing down from the app.jsx component we're drilling that into if you like drilling that into the um header component so that when the user clicks the logout button it runs the relevant logout code so we actually haven't created this endpoint on our server so that's the next step cuz at the moment that's going to nowhere but if that serverside code did exist we're posting the users ID the relevant users unique ID to the logout endpoint and the logout endpoint handles the logout functionality on the server but let's create the server side code the relevant server side code for logging out okay so I'm actually just going to just paste that in from the code that I've got so let's go to user_controller and I'm going to paste in the code from the prototype code that I've created in preparation for this course just to save time and we'll just briefly look at the code here so we've got a function called logout handler that hooks into jinggonic so that we can expose an end point that maps to this function so when the user goes to that endpoint or when the client calls that endpoint this function is called and the relevant logic is applied so we've got a logout strct here local strct that includes uh JSON data the user ID it maps to that so this user ID will will store the user ID passed in through the body of the post request to this logout handler function and then we're binding to the strct with what's passed in through the body the user ID which will be passed in as JSON data through to this function when the relevant endpoint is accessed and it's uh accessed via a post request a HTTP post request so we're handling any exceptions here then we're just logging it to the screen for testing purposes we're logging it and then we're updating the relevant tokens to an empty string here because we're logging out we don't need these stored anywhere on the server the tokens so we're logging out so that's why we're doing that and handling any exceptions and then we're setting the token to expired minus one to invalidate the token um the HTTP cookie so it cannot be used anymore in subsequent requests and we're doing the same for the refresh token and then we're just passing back a message to the client with a status response HTTP status of status okay and a relevant message logged out successfully so now the next thing of course we need to do and we can do it in unprotected routes is we need to map an end point to the logout jinggonic function so it's so it's an unprotected route and this is just this unprotected route has this and the path log out so this is the end point log out and then we of course need to map it to the appropriate genonic function which is log out handler log out handler so let's go to the client and see what's happening there so we're posting to the logout we're posting the users ID it's returning us HTTP status of okay hopefully and if all has gone well we're setting or to null so we're setting that global orth variable to null and we're removing it from local storage okay so let's test that out let's see if we can log out of the system now so let's run the server go run dot and let's run the client let's first clear the screen here and let's go mpm rundev to run the client let's copy this URL to the browser window like that okay great let's authenticate as goth jenkins hotmail.com password one dot okay log in we still need to do something about this but it has said hello goth but it's in black so we can't see it against the black header here but we have logged in successfully and we can go to recommended that proves that we are logged in successfully and then uh we can log out by pressing this button which is not working okay so let's just see what's going on here oko logged out reference Axios client is not defined at handle logout okay oh I see okay no it's just an importation problem there okay so let's fix that i'm just going to go to the client and quickly fix that we're not importing Axios clearly here okay okay so we're trying we're trying to call the logout endpoint using the Axios client but we haven't imported the Axis client here so that's what's happening there and it's easy to remedy this problem with this line of code here we're just importing the Axios client so we need to import the use or hook here like that and make sure that we set a variable the state variables or and set or retrieve the state variables or and set from the use hook like that so this should actually be hook not hooks let's change that there mpm rundev okay copy that URL to our browsers login as Goth Genkins okay are we logging in here everything's working and let's see if we can log out brilliant so our logout functionality is now working brilliantly that's great okay so what I want to do here is for the use private hook I'm going to replace this code so now you'll see the significance of using a private hook here to get the to use this use Axios private hook where we're creating a completely different access object to the one we're using for protected for unprotected endpoints here and of course we're using this hook code here that creates a completely different Axio object and the reason I'm doing that is so that we can refresh the token which will keep the user logged in but this code is super involved and I'm not going to do it line by line because you'll see it's got there's a lot here involved but I will give you a brief explanation of what's going on by all means please look at the code on GitHub and copy and paste it into your own applications and go through it line by line yourself to get an understanding of what's actually going on here briefly I'll explain briefly what's going on here so for example let's say the user has left the user's browser open and the token has timed out so what's what's actually happening here is this code is going to receive a an error a 401 error here so what it does is it puts that request here within a failed queue so it puts it within a queue here and then this code here traverses that queue and tries it again once a new refresh token a new token the token has been refreshed so this is why we have a refresh token the client you know once the access token times out so it gets the appropriate error sent back to the client the token is timed out this client makes a further post request to this refresh endpoint which we haven't yet written we'll write that code in just a bit and it gets given a new token so the access token and the refresh token are both refreshed on the server and passed back to the client and then the client then this code here that traverses the failed queue will then try again with the new refreshed token to access that protected endpoint so the tokens are refreshed so in other words it doesn't just log you out the the client is still remains lo the client still remains logged in and our code here just automatically refreshes the token on the server so that the client remains logged in it doesn't just log you out automatically or present you with an error that your token has expired it refreshes the token for you and we've even got code here that if you've left the token to expire in other words you've just left your browser open and the token has expired and the refresh token has expired then it logs you out and presents you with the login screen so you have to reauthenticate but you can control the time on the server the time when the tokens expire but this is what this code is doing here so if your token has expired and you try to access a protected endpoint it won't just log you out it will make a post request to the server to get to refresh the tokens so you will remain logged in and on the quiet it's refreshing the tokens and then a subsequent then it will process the queue here and a new call to that endpoint that you tried to access where the token expired will be made with the new refreshed token so that will go through and so you won't be disrupted on the client and this makes for an excellent user experience so you will remain logged in until you actually manually log yourself out which will invalidate the tokens so this is how this code here is working but I urge you to go through the code the logic line by line and try to understand this code so that's what this code is doing here it's refreshing the token if the tok if the access token expires and keeps the client logged in so that the client has control over when the client logs out okay so now you can see that we're making a call here to this refresh endpoint we haven't yet created that refresh endpoint so I'm going to go to the server side code and create that refresh endpoint within the user controller file well we're going to create the handler function within the user controller file and map it to an appropriate refresh endpoint so again I'm just going to copy this code in here to save time but please by all means go through the code yourself to understand what is going on okay here so let's copy this function handler code in here and you can see what's going on here we're getting the refresh token from the cookie here we're validating the refresh token so the refresh token is automatically being passed in via the HTTP only cookie that we're setting when the user logs in so the token has what's happened here is the token has expired on the client and instead of the user being logged out when the user tries to access a protected endpoint the code automatically makes a call to the refresh endpoint and this logic is run this jinonic handler function is run and the refresh token value is extracted from the relevant HTTP only cookie here and we validate the refresh token here and we make sure the user ID that is also sent through with the refresh token it will be within the refresh token encoded within the refresh token so we can access the user ID by extracting that user ID from the token itself so it's being decoded the claims are the claims that are encoded within the refresh token are being decoded and read here so we're able to read the user's ID from the claim here and find the user within the database um and then we can generate new tokens so we're generating a new refresh token and a new token and then we are saving those relevant And then we are saving those tokens to the relevant HTTPON cookies here and the tokens have been refreshed so the user remains logged in when the user accesses a protected token and the access token may have timed out our client code automatically makes a request to this logic here via an appropriate endpoint and the tokens are refreshed and and saved to the relevant HTTP only cookies so the user can remain logged in and can access those endpoints so it's up to the user to log out manually so that makes for a much better user experience but if the refresh token and so if the token if the access token and the refresh token have timed out we've got this code here that actually logs the user out properly in other words it removes the local storage setting the or setting bear in mind this user local storage setting will no longer store the tokens so it will store information about the user but not the tokens so there's no security risk in storing the the sort of the first you know the first name the last name and information like that so the tokens are no longer stored here but we remove the user information and we set the orth global or state variable to null so we're logging the user out if the access token and the refresh token have expired then we log the user out with our code okay so let's go back here user controller okay have an issue there we need to close that off there save that getting a problem there user dot user ID user collection i think we called it user collection sorry let's change that to user collections cuz I'm copying and pasting the code validate refresh token okay we don't have validate refresh token we haven't written that code yet clearly undefined yeah so we haven't written that ref uh validate refresh token function yet user do user ID saying user do user ID is undefined so we're mapping that user strct there so user do user ID doesn't like that there um just going to look at my what my model looks like here yeah user ID okay so we've included that as a capital D okay so that's the reason so I'm just going to make that a capital D to solve that problem there um validate refresh token we need to write the validate refresh token i've already written that in the prototype so I'm just going to copy that validate refresh token it's very similar to the validate token method but obviously it's meant for to validate the refresh token so I'm just going to copy that in there let's go to our token util file here and paste it in there okay um we have we're not reading in the key clearly there so we need to make sure we're reading in Oh secret refresh key okay we are reading in the secret refresh key there but we referencing a different name there so secret refresh key let's update that there and that now looks good and all we're doing here is writing standard Go code here to validate the refresh token in much the same way as we've done to validate the access token here it's pretty much it's this code is very similar okay great let's go back to user controller what's the problem let's go to problems ah user ID so that user ID there also needs to be changed to user capital ID like that okay fine no more problems now that we've written our refresh token handler the next step is to map an endpoint to this refresh token handler function gingonic function so this will also be an unprotected route and let's include the relevant post request to the refresh endpoint there the relevant code that maps a post request to this refresh endpoint to the relevant gingonic handler function okay I think we're ready to test that now let's just double check that I haven't broken anything we're not going to do a detailed test on that refresh functionality but you can by resetting the various timeouts you can trigger a a uh access token you can trigger the access token to expire by appropriately setting its time out and then you can test the refresh functionality yourself okay and I'm going to run the client dev npm rundev like this okay let's copy that to the to the browser okay big screen that looks okay let's log Gu Jenkins in hotmail.com password one exclamation point great okay so we're logging GTH in obviously we need to do something about that no point in having Hello Goth in black against a black background so we'll do something about that recommended so it's all working fine and can we log out yep we can perfect so we're in good shape okay great and then to do with the authentication on the front end and how it affects the front end and logging out there's one more test I'd like to do firstly actually I need to run the code again run dot npm rundev i just want to show you something at the moment why the login functionality isn't ideal just yet so firstly I just so I want to show you why the login functionality isn't ideal yet so if I log in uh okay so let's log in as G Jenkins again why not stick to the same user Jenkins atotmail.com password one exclamation point so we're logged in and we can access all the relevant endpoints all is just fine and dandy and wonderful great but now what happens if we refresh the browser window look at that logs us out so we've we can no longer access our protected endpoints so that's all so that's a huge inconvenience if the client if the user just refreshes his browser and just gets automatically logged out if for some reason the client decides to refresh the browser look what happens and we're still you know we don't want the user to be logged out at this point we want the user to remain logged in even when they refresh the browser so we can actually use the to to resolve this we can actually use the orth information that gets sent down from the server when the user logs in and we can we are currently saving that to local storage and we can use that to check whether the user is logged in or not so when the user if the user refreshes the browser the user can remain on the recommended page and not be presented with the sign-in screen so let's write the code to resolve this issue okay I'm just going to cancel out of that clear the screen okay we can keep the server running we're not going to create any server side code we're just going to amend our code on the client to resolve that issue so to do that you can see what's actually happening here first let's go to orth provider we've got the orth provider component here we're going to amend the code within the orth provider component but what's actually happening here is if we go to required or here you can see that it's checking the orth state variable it's retrieving it from a global store within the context so it's retrieving the or state variable it's checking it here if the orth is truth the it outputs the expected page that the users navigated to so in this case it would be the recommended page but if that or is falsy then it navigates it explicitly navigates the user to the login route here which is what we want if the user hasn't yet authenticated but if the user has authenticated we want the recommended page displayed to the user so to resolve this issue when the user for some reason decides to refresh the browser and then it presents the user with the login screen we don't want that if the user has authenticated but we do want it if the user is not authenticated because then the user can authenticate through the login screen and the login functionality but if the user is just refreshing the page but is in fact being authenticated we want the recommended page presented to the user we don't want the user to be navigated to the login screen we're setting this local storage value here with the response sent back from the server when the user logs on which contains various information about the user now what's important to note here is that we are no longer sending the tokens back for security reasons that's being handled by HTTP only cookies so but we are sending information about the user that is not so sensitive like the user's first name last name email that's okay to store locally because it doesn't matter if some nefarious code comes along and reads that information it's you know it's not going to allow them to access protected endpoints like the token would because the token is like a key that can be used to unlock protected resources but we do want to store information about the user returned from this login endpoint once the user has authenticated and we are storing that in local storage within the browser we're storing it statically so whether the user if the user refreshes the browser the context information is actually flushed out of memory but this user information that's saved to local storage will be stored statically within the browser so we can still read that information from the browser but we can't read that state information after the users refresh the user's browser we can't read the state information here in the orth variable and the orth state variable that will be cleared from memory so we won't be able to read that so that is the reason why this code here is then navigating us to the login screen once the user refreshes the browser because that orth state has now been cleared so it's set to null so this code is running correctly but that's not what we want uh for the user's experience to be so to resolve this let's go to orth provider here and the first thing we want to do is create so firstly I'm going to create a loading state variable and you'll see how we use that in just a bit and then we need to include this code here and you can see what we're doing here we're reading that user data from local storage which is stored statically in the browser so this is the way we can get around um if the browser refreshes the user's browser if the user refreshes the user's browser that O information is lost pretty much even if the user's been authenticated so that's currently what happening but now we're actually reading that information from local storage and if that is successful we're resetting that state variable so after the user refreshes the browser we're reading in that information from local storage and resetting that orth state variable which is accessed now globally which can be accessed globally through the use or hook that we created so that's what we're doing here so when this loads again we're reading that information from local storage which is stored statically so the information when the user logged in is not lost so this is how the user can remain logged into the system even even if the user refreshes the user's browser and the orth information is lost we can read that information in through this setting in local storage okay and the next thing we need to do is include this use effect so when the orth state variable changes it runs this so if author is truthy we can set the item in local storage and if it's not truthy if it's falsy we're removing that from local storage there and now we we're handling this centrally within the orth provider component here okay and we need to import use effect because now we're using it twice within this component here the or provider component okay okay we got two use effect hooks running here one that executes when the component loads so it's reading in the users's information from local storage and the other use effect only runs when the orth state variable changes and that ch will change when the user logs in and logs out so it'll be set to null when the user logs out and it'll be set to the users's information when the user logs in the relevant information but it won't include the tokens so that our application remains secure we are handling the tokens through HTTP only cookies okay excellent so now let's go back to the login so now you can see here where where we're adding it to local storage we're now doing it centrally within the orth provider um component so we no longer need to do that here because this use effect will run when changes within the login component so when we set off the this use effect will trigger if you like and this code will run so if orth is truthy which it will be when the user logs in we're setting we're adding that information to local storage and when we log out conversely when we log out so let's go to the app.jsx file here when we log out here we are actually removing it explicitly but we don't need this code because now that code is centralized centralized with an orth provider here so we're removing it when the orth is set to null so it changes this use effect runs it goes ify then set it but if it's falsy if it's been set to null and it's changed from truthy to falsy in other words it's been set to null the user's logged out we remove it from local storage here and that code now is central is centralized within the orth provider component and we need to pass down the loading state variable so we're setting loading to true here because when we're setting it to true by default and then when the use effect hook runs it's going to be loading and then if and then once it runs through this code the loading state variable is set to false whether an error occurs or not it's still set to false so that loading indicator will disappear okay and then the next thing we need to do is go to required orth now we need to see exactly what required orth does first so let's go back to app.jsx you can see here we're wrapping these protected roots within the required orth element here so this required orth protects these roots on the client so let's go to required or and see what's happening so you can see if orth so if the user is authenticated you can go to the protected route the or the user will see the protected route that it's trying to access else if that is null the user's explicitly navigated to the login screen so the user needs to authenticate so we need to include and this logic is fine by the way that's perfect so we actually need to include this code here so we're going to include a loading indicator this our spinner that we created earlier on so we need to import the spinner there so we're going to show a loading indicator if that loading state variable that we are retrieving from use or the use or hook which engages with our context that we've created within or provider here see we've created this loading state variable here that's being set appropriately within this functionality that reads in the users's information and then sets the orth state variable appropriately um so we go back to required or here so if it's loading we want to display the spinner and we're importing the spinner there great okay we're now retrieving the loading state variable which is now set globally within our context and this logic is correct we don't have to change anything else so I think that will now work and we're no longer setting that um local storage variable once the user logs in because we're setting it within the lo login component because we're setting it now centrally within the orth provider and remember the orth provider component or element wraps all the roots so that's that's how we can pass down the context that's how the context is available to all of our components within our application which includes our protected roots where it's needed for for example with the recommended component here okay one other thing I want to do is within orth provider I just want to put this try to to this try code here to encapsulate all of this functionality here okay so let's just move that down outside of these this curly bracket there that makes more sense okay that looks all right okay so we also we want to catch any exceptions like when it's reading the local storage setting we want to catch that if there's a issue there and of course we want to set the loading indicator to false whether an error occurs or not when processing this code here which reads in the user information pauses it because it'll be in string format when it reaches reads it from local storage and turns it into an object here and sets the orth state variable pm rundev okay let's go to the relevant URL here recommended okay let's log in like that let's go to the recommended page here and let's see if if what happens when we refresh this page brilliant okay there we go so we're refreshing the page by selecting the address and pressing the enter key and now we're remaining logged in so this remains in the log in state and we've still got Hello G here in black i need to sort that out um but yeah so that solved our problem we're now able to refresh the browser window and remain logged in so that makes for a much better user experience you don't want the user to perhaps inadvertently refresh the user's browser and be logged out of the system that doesn't make sense so but we want the user to be able to explicitly log out on their own terms which is working now great so the user now needs to reauthenticate go genenkins@hotmail.com password one exclamation point and now the user just remains logged in no matter whether the user refreshes the browser or not okay and so that works as expected as you as one would expect that to work hello go that needs to be in white I think um so I'm going to just sort that out um so I'm just going to add a class to we can just leave that running there go here let's go to the header component and where it says hello goth let's just include these classes to style it better and that should propagate right through and there it is hello Goth excellent so we've got now Hello Goth appearing appropriately in white on a black background that looks much better excellent so that's all working for good measure let's just try and refresh this a few times and we're good we're remaining logged in as Goth Genkins okay great so but we can log out whenever we choose so let's log out now and we're logged out and you can see we need to reauthenticate if we want to go to the recommended page now okay so we're good there brilliant i'm really happy with that okay now the next thing I want to do is create kind of a substitute for the streaming functionality and in fact all we're going to do is play a YouTube video a trailer for each of the videos which just substitutes for the streaming functionality so we're going to bypass creating actual streaming functionality and just substitute in playing the trailers in a React component called React Player so the first thing we need to do is actually install React Player and it's very easy to do okay so now we want to be able to stream our movies and we're not going to actually create an actual actual streaming functionality in its place we'll just run a trailer that exists on YouTube um within a component called react player so we're going to use a third party component to actually play the movie the relevant trailer per each movie if we look here in compass we can actually see that each movie each movie contains a field called YouTube ID that contains the ID of a movie that has been published to YouTube and these particular videos are the trailer for the relevant movie document within the movies collection so that's how we are able to play a video pertaining to each of these movies so we're not actually streaming the movie we're we're in fact playing the trailer for the movie which exists on YouTube and we're going to use a component called React Player to help us create that functionality so let's go into our code so I'm going to start by creating a folder within the components folder and I'm going to call this stream with a small s here within stream I'm going to create a file called stream movie like this.jsx and I'm actually just going to copy in the code for stream movie but firstly we need to install a component called React Player which will be leveraged to play the actual movies trailer within the browser so let's go to the client terminal window here and let's install React Player npm install React Player now if we were to run this it would install the latest version of React Player and I don't want to do that because there was some breaking changes I found with the latest version so the one that I know works is version 2.6.0 so to install a particular version you can just include the at symbol like that and then the relevant version 2.1 6.0 zero and let's press the enter key and it will install the appropriate version of React Player so I'm going to just copy in the code here for this component and all it does is play a movie a YouTube video which is the trailer for the relevant movie you can see a parameter gets passed through with the YouTube ID we're reading that parameter and we're including that as part of the URL and then within React Player the component we're importing here it plays the relevant trailer for our movie okay so then we want to include it within the protected roots here within app.jsx so let's copy this code i'm just going to copy this code in here that will point to our stream movie component we need to of course import streamov within the app component we can do that with this code here so we're importing it there and we're mapping our root path here to the relevant stream movie component there and you can see that it's got a YT ID parameter so this is how we're passing in the YouTube ID to the relevant trailer to the stream movie component okay and then we need to go into the movie the movie component here and you can see offscreen I've already created the code for this here we've got a key here that points to that uh makes this div tag unique this is of course the MongoDB unique identifier for the document that we're using here as the key for this div element here and then within here we've got with we've imported the link component from React Rout and we're wrapping the card each of the movie cards within the link element and we're pointing through this two attribute we're pointing to our stream component and we're passing in you can see here we're passing in the YouTube ID so that the relevant trailer can be played within React Player and that's the component that we recently installed and that we're using we're using that third party component to help us play the movie within the browser okay excellent so you can see we're closing the link down here that's wrapping the card within a link and that should be great we should be able to play our trailers for our movie so let's make sure that the server is running so let's run the server go run dot and let's test to see if we can now play the trailers for each of the movies by clicking on the relevant movie cards okay so let's run the client mpm rundev like that and let's copy this to our clipboards let's open the browser window and let's see what we've got here we've got a problem straight off the bat okay right so streamoviecs failed to resolve okay super JSX okay ah I didn't include the relevant uh stylesheet okay so I just want to include a stylesheet within the stream folder here it's a scoped stylesheet so let's create a new file called stream movie.css like that okay and the CSS code is very basic it's just ensuring that the movie takes up the movie screen react player takes up 90% of the viewport and it's going to be responsive on smaller screens with this particular style here you can see here we're setting the class name to that style okay and now I think we're ready to test out the functionality for playing a movie within React Player okay and let's go npm rundev like that okay let's copy that to our browser window and let's test our code let's log in as G Jenkins remember it's a protected route because we only want logged in users to be able to utilize the streaming functionality provided by our Magic Stream web site so pass word one exclamation point and let's log in as goth Jenkins and let's try run one of our movies excellent look at that i love that stuff okay and now we're good we can run our movies here he's the best and this is just not really ready for a relationship and no one okay so clearly it's streaming our movies well it's not the warden i didn't say since you ask the outside was an honest man okay and we're now able to to simulate the streaming of our movies just by playing the trailers on YouTube blue i am a bag great i'm really happy with that and there's only one thing left to do really before we've finished and that's brand our web application and I've prepared a a little icon that I actually generated through chat GPT um for the magic stream branding and then we just need to put it in place so we want to place it here within the assets folder that I mean the the actual icon that represents Magic Stream so I'm going to So I've got my icon here that represents Magic Stream i'm just going to copy this from the prototype code that I created to this folder here so I'm just going to go reveal in file explorer going to assets and paste it there and it should show up there there it is magic stream as I say I just generated this through artificial intelligence and now we're going to place it with appropriately within the header so where it says brand here or somewhere here uh there brand we want to include an image here just above where it says magic stream let's include the image okay and you can see we've got this logo variable here so we need to import the image from our assets folder and we can do that with this line of code here so we're importing this magic stream logo into the header component and we're just we're branding our web page so and then we want to include this also in our registration dialogue so let's go to the register component here so we want to include our logo here also within register okay with this line of code here so we need to make sure we're importing the logo and we can do that with this line of code and then we want to do it again for the login form the login dialogue the login form the login screen so let's import logo there and then include the image within that login form to brand our website great so let's test what we got mpm rundev let's see if our website is now branded appropriately okay let's go there let's run that look at that we got Magic Stream there now so our website is appropriately branded and let's go to login there it is magic stream register let's register a user and I think that looks pretty good and now we can make use of our magic stream website and we can taste the streaming those are two boys okay that's looking good so if we log out we shouldn't be able to stream these movies because we're not authenticated so let's try that so if we click on that movie we need to authenticate before streaming the movie whoops and let's log in brilliant so that's all working knives out let's try that daniel Crane ladies and gent so that's all working perfectly i'm really happy with that and that is basically our functionality already in place and we can see the admin reviews here brilliant and we can run our movies there's one other thing I'd like to do and that's just putting in a play button just in the middle of the movies just for effect so we can do that quickly and that'll be it then I think our functionality is all in place and you can see all the code on GitHub on the relevant repository which has been included below in the description so we want to include on the movies on each of our movies a play button so I'm going to include a stylesheet here new file movie.css and let's include the code for a transform effect when the user hovers the mouse pointer over the movie and it also includes a play icon that will be situated over each of the movie cards in the middle of the movie image okay let's go back to movie.jsx and let's implement the code for that and in fact we need to in order for that to work we need to install Font Awesome okay okay so we want to include a play icon in the middle of each of our movie posters so we're in the movie uh component here the code for the movie component and we're going to use Font Awesome to include our play button icon within each of the movies so we've already created the CSS for that and we've got these classes here that we're going to leverage one to create a hover effect over each of the movie cards and the other to place this one here play icon overlay we're going to overlay the play icon within the movie card so each of the movies will have a play icon in the center of where the poster is in the movie card um so let's go back to the movie component code okay and we want to firstly install font awesome okay so we want to install two font awesome packages let's install the first one npm install at Fort or some slash free dash solid dash SVG dash icons press the enter key okay brilliant that's installed that great so we've got free-solid- SVG icons installed let's clear the screen and install the second package we want so npm and install for oops at fort awesome for slash react dash font or some like that let's press the enter key brilliant so we've now got our font awesome packages installed and let's import them up here so that we can leverage their functionality so import And it's font or some icon from open quotations at fort awesome free solid SVG icons like that okay excellent and then we want to I'm just going to duplicate that and we want to import a particular icon called FA circle play that's the icon we want to import and we need to import it from oops sorry this one should be react react dash react dash font awesome like that and this is the one we want for free solid SVG icons we're importing a particular icon from free-solid- SVG icons like that and then we want to place that within the center of our of each of our movie images so to do that we're including a span element below the image element here and then we are handling the actual placement of the FA circle play icon with within this movie- CS movie.css file here with this CSS code here if we go back to movie you can see that we are leveraging that class that CSS class to place the FA circle play icon within the middle of our images and there's other functionality here for hovering our mouse pointer over the card and you'll see that we've created a hover effect a custom hover effect over each of our movie cards when the user hovers the user's mouth pointer over each of the movie cards to highlight that that particular movie is in focus i think that looks pretty good but one thing we haven't done is we haven't imported our movie.css file so we got to do that and we can do that with this line of code here okay great let's run our code mpm rundev our serverside code is already running and let's copy this URL to our browser window close it down here like this okay excellent so that's worked perfectly our play icon is appearing within the middle of each of our movies but when we hover our mouse pointers over the movie that currently doesn't seem to be working and that is because we haven't applied a particular class so let's leave that running and we need to go back to our code here and which class are we missing we're missing a particular class um and it's movie card that we're missing and we move missing movie card here see we got our card Bootstrap icon which turns this div tag into a card and we need to apply our custom CSS code which is in the movie.css file to this element so movie card and that should include the font that should include the hover effect now within our code let's see if that works oops go here and there we go and we've got our hover effect now working so we've got our play button signifying that we can play we can stream the actual movie of course it's just going to play the trailer of the movie as a substitute for that streaming functionality so let's see if that works so if we play that there it's prompting us to authenticate because it's a protected route we can only stream the movies if we are authenticated genkins@hotmail.com password one exclamation point log in and it should take us to the movie and stream it for us i suspect it had no suspects well that's looking pretty good jack Reacher okay good old Tom Cruz he knows what I did nothing to lose okay let's go to the recommended section we got Unforgiven fantastic movie maybe I ain't bad man no more we're 11 oh they wouldn't want to come looking for kill them cowboys and that's it basically we've got all of our functionality now in place and that is awesome so let's see what a review looks like here we can review our movies we can Well if we're just a user we can view the reviews but we can't update the reviews so let's log out and log in as a an administrator bob Jones@hotmail.com password one exclamation point we logged on as an administrator this movie wasn't too bad actually should be this movie it's not very good okay and we can actually change this review this movie isn't too bad let's just say this movie this movie was absolutely fantastic exclamation mark submit the review excellent so it's changed the sentiment from good to excellent this movie was absolutely fantastic and we can even watch our movie if we want there we go and that is looking really good i'm really happy with that i don't think I've ever been this hung over what's on your arm the face yeah that's brilliant and that's our application done let's log out so we've pretty much coded our full stack web application we've created our data store using MongoDB we've created a web API component written in Go that runs on the Jinggonic web framework we have created a UI client using React so that the users of our application can benefit from the high-erforming user interactive real-time UI responsiveness provided by React but there's no point leaving our code hanging about on our local machines it's great for testing and development but we are eventually going to want to share our application with the public so in this part of the course we are going to publish our application to various cloud platforms the components of our application are going to reside on three disparate platforms so this is an example of a distributed scalable application we have also employed cyber security techniques so that our client and server components can communicate with one another across domains using HTTPS we are firstly going to publish our MongoDB database to the Atlas platform we'll then seed our deployed MongoDB database on Atlas with the data we have stored on our local machines through the use of MongoDB database tools mongodb Atlas is MongoDB's fully managed cloudnative database service designed to simplify the deployment management and scaling of MongoDB in the cloud we are going to publish our ginonic web API component written in Go to Render render is a modern cloud hosting platform that provides developers with a way to easily deploy and scale web applications APIs static sites databases and background workers it's often described as a middle ground between traditional infrastructure services like AWS and fully abstracted platforms like Heroku one of the reasons I chose render is because it still has a free tier that we can use we are going to deploy our react client code to versel versel is a cloud platform optimized for front-end development especially for building deploying and scaling web applications with modern frameworks like nextjs react view spelt and more it focuses on speed simplicity and a smooth developer experience versell also provides a free tier that we can use great so when we created the code for our application we configured our server and clients to communicate with one another via HTTPS even if they reside on two separate domains so HTTPS ensures that the data exchanged between client and server is encrypted on the wire which is part of our overall cyber security strategy we are using token-based authentication and storing the tokens within HTTPON cookies so that for example malicious JavaScript code cannot be injected into the client and steal the tokens so we have protected the client from XSS attacks cross-sight scripting attacks as discussed the tokens are like keys and keys can be stolen and used to gain access to protected resources so using HTTP only cookies is another significant part of our overall cyber security strategy so we have built a robust and secure full stack web application where our client and server can reside on two different domains on the internet and communicate with each other securely so let's get going we're going to start by creating a MongoDB database in the cloud and we're going to use the Atlas platform for this purpose so firstly we need to create an account on Atlas so we need to navigate to https/mongodb.com and then products platform atlas database great and I'm going to sign in i've already created an account if you haven't yet created an account please create an account um you can just click this get started button here i'm going to sign in i've already created an account with my Google credentials so I'm just going to press Google here signing me in automatically excellent and then it's very straightforward um and if you haven't yet created a cluster you can create a free cluster so you can use a free tier on Atlas and create your MongoDB database um so the first step to doing that is to hit this create button here to create a cluster and I'm going to select AWS here as my provider and then this free option here and I'm going to name my cluster Magic Stream Movies like that um going to choose America here north Virginia and America free okay free forever your free cluster is ideal for experimenting in a limited sandbox so this is really for just for testing purposes and we can just create deployment now so let's click the create deployment button here okay excellent okay and then access your Atlas data using MongoDB's native drivers for example Noode.js Go etc and we want Go to be able to uh access our Mongo uh DB database so I'm going to click this option here Node.js driver i'm going to select go from this drop-own list here okay and then we have our connection string set up here okay so we need to replace DB password with the password for the Gavin London Database user okay so we can copy this to our clipboards we've got our connection string let's copy that to our clipboards we're going to be using that within our application to connect to our deployed version of our MongoDB database and of course we need to replace this with the actual password which we haven't yet created okay great so we've created our cluster here but we need to include a password within our connection string you can actually add users to your cluster so that you can include that in the connection string that you'll use from the application to connect to the relevant database our magic stream movies database and you actually do this at the cluster level on Atlas so how to do that is you go to this database access option here and you can either add a new database user but what I'm going to do is just edit my Gavin Lon D user here and add a password that I want here so I'm going to go edit password okay and choose your own i'm just going to use just for simplicity let's show it password i'm just going to include password one here it doesn't have to be uh super secure we're just testing it we're testing the deployment so I've just chosen a very simple password password one like that okay and I'm going to update user so the next thing I want to do is actually seed the database with the data that we've got stored locally in our local version of the database here Magic Stream Movies and we can actually do that through the command line prompt through MongoDB database tools so you firstly need to make sure that you've got the relevant tools installed and you can actually do that by navigating to this URL www.mongodb.com try download and then data face hyphen tools like this press the enter key okay and then it's detected the site's actually detected my platform and it's just a case of downloading the relevant MSI file and running the installation to make sure that you've got those tools installed um which we're going to use so that we can deploy or that we can seed the database that we've got running on Atlas um with the data that we've got running locally and you can verify whether you've got the tools installed by typing mongod dump mongodump- version like this okay so I do I've got mongod dump installed so now the first thing we want to do is find out where this is the path to this location here so we can rightclick that there show connection info okay and we've got local host 2707 so we need that connection there i'm just going to copy that there so we got a record of it there so local host this is the path this is the connection string to our local databases MongoDB databases so we first want to make basically we want to create a we want to kind of export it we want to create a copy of our database and we can do that using mongod dump so we're going to do that with this code here mongodump MongoD dump and then dash dash URI the d- URI option equals to and then within quotation marks go mongo db colon slash slash and then we want localhost colon 27 1 27 7017 017 should I say so localhost colon and the port numbers 27017 and then we want the name of our database which is the name of our local database which is magic stream movies so include magic stream movies like this and then a dash dash out option equals to and then dot slash dump like that so we're creating a copy of this data if you like from so we're creating a copy of this local database with this command press the enter key and now we want to update uh the database we've just created on Atlas with the data that we've just dumped out here on on our local on our local machines okay so to do that we go mongo restore dash uri equals to and then we need to include this connection string here so I'm going to copy that and paste it there and we need to replace this password with the password that we chose through Atlas so let's replace this here with pass word one like that okay and let's see if that works okay and that looks pretty good i think that's worked 37 documents restored successfully zero documents failed to restore so now we can actually go to our server side and see whether that has worked whether we've got data in our database um so view all projects project zero let's go to our project here and we should be able to browse collections now so we can click the browse collection option here and see if our data has been restored within the database that we've got on running on Atlas look at that that's exactly what we wanted so we've now got a copy of our database that we created locally we've now got that data seeded on the database that we created remotely in the cloud on Atlas on the Atlas platform brilliant so we've deployed our data pretty much great so we're going to upload our serverside code our Go Jonic code to render the render platform but firstly in order to do that we're going to push the code that we've got for the client and server to the to a GitHub repository so if that's the first step we need to create a GitHub repository and then we're going to push the code to that GitHub repository so let's go on to GitHub so this is how you would do it so once you're logged on to your GitHub account to do that you want to create a new repository so we go new repository and we're going to call this one magic stream movie like this okay and it's available so we can use this name magic stream movies and we're going to I'm going to make this private for now but I'll make it publicly available once the course has been released okay so let's do that let's create the repository there and uh the thing to note we've created this magic we've created the repository named magic stream movies is that if you go to go.mod and you'll recall that we created when we initialized our server side code our go code we created the the module based on this name here so you can see that this first part of this path here is is the path to the repository that we've just created so this path would be similar only it would uh contain your GitHub username here within the path okay and now we want to push our code to GitHub so firstly let's open an appropriate git bash uh terminal window like this we're going to use git bash to uh create our repository and push our code files to GitHub so first thing we need to do is initialize our root folder that contains the client and server code as a repository and to do that we can type git init like this so that's the first step and it's all turned green then then I want to make sure that certain files like for example eenv files are not included in the upload we don't want to upload those to to GitHub so to do that let's create a git ignore agit ignore file touch and then dot get ignore like this and we've created a dot get ignore file here and I've prepared offscreen the co the settings I want in the get ignore file here like this okay so you can see it's grayed out there and env there because we don't want those included and of of course all these node modules we don't want that included within the within the within the the remote repository so it's already Visual Studio has already detected these settings that I've included here okay and that looks good and now type get add dot so we're adding the code files the relevant code files to our repository great and then we want to I'm just going to clear the screen and then we want to commit that action so get commit and this is at this point we can include a message for this particular command associated with this uh commit and I'm just going to type initial commit of full stack magic stream app like this okay that looks good let's press the enter key okay that looks good excellent and now we want to push those files to our new magic stream movies folder that we just created on GitHub so to do that we can run these commands so the first one is get remote and then add origin so we're adding from origin and then we can include so this path that we're going to include is based on this base path here which includes github.com and then my GitHub username so obviously it'll include your particular username um in your particular case so https slash slash and then let's include we need to include this path here okay cuz that's where we want to upload our code to on GitHub and then let's press the enter key brilliant that looks good and then let's make sure that we're pushing to the appropriate branch which is the main branch branch dash m main like this press the enter key okay all good so far and now we're pushing our local code to GitHub so the remote repository so get push dash u origin main and this is the main branch we're pushing to okay and let's press the enter key to do that and hopefully all goes well well that looks good excellent so now let's go on to GitHub and see if that code has been successfully pushed let's refresh this excellent so our code is there and do we have environment thev file there no we don't and that's what I want we only want the code files there and we don't have things like the node modules pushed to GitHub we don't need those there and we don't have thev file here that's brilliant okay so now we're ready to publish the server side code from GitHub to render so the first thing we need to do is go to the render platform render.com excellent and I can go to the dashboard here like this but I need to sign in so I'm going to sign in using my GitHub credentials i've created my account using my GitHub credentials so you can do that too i guess it makes it easier to publish from GitHub if you do it like that okay so the next step is we can click we can select this option here deploy a web service okay and then let's select our repository it's already detected all my repositories in GitHub so Magic Stream Movies okay us we want to keep it on the east of US so Virginia would be ideal that's where our MongoDB database resides okay so we want to include a root directory here so for the root directory it is server magic stream stream server i believe it's movies let's just double check that so if we go to our code here yep server magic stream movies server so magic stream oh magic stream movie server so that's not right magic stream movies server like that so that's from our root from the root of the magic stream movies repository we want the root directory where our commands are run to be run from this directory because that's where our go code resides and we want the free tier there so all our settings look good now and I think we're ready to deploy you know ah very important step we're not using the environment variables so we need to add our environment variables here okay so we're going to do that step by step now so let's go to our code here let's go to thev file here which has been grayed out appropriately so let's just start database name okay so database name and that's magic stream movies like this got that there now add environment variable MongoDB URI and this is going to be different now to the one that we've got here the local one um so let's so let's select let's go back there MongoDB URI and in fact what we want is this that we copied from from Atlas it generated this connection string for us so this is the connection string we want we don't want a local connection string so let's copy that and paste it there we got database name and we've got the relevant connection string uh and we need to change this password here to password one that's what we changed the password to for our MongoDB database on Atlas so it's password one just to keep things simple this is obviously just for testing purposes um for your actual production environment you obviously want to select a more secure password okay and then let's go back here secret key and we're just going to keep it your secret key environment variable secret key like that add another environment variable secret refresh key okay secret refresh key and we just keep going through it like this until we've copied all the settings all the environment variable settings to the render platform like this okay back there base prompt template and of course that's prompting open AI okay and we need to copy all of this here right to the end okay copy that go here paste it in there add another environment variable what's the next one called okay and this is our open API key that we created on the open API the open AI platform so this is the open AI API key that we created on the open AI platform let's include that there let's go back here and let's copy the relevant key here i will deactivate this key once I'm finished with this course so you will need to obviously generate your own key if you want to try out the open AI functionality okay let's go back there paste that in there okay let's add another environment variable and recommended limit nearly there back there recommended limit and it's just a value of five like this and we know we got one more environment variable to configure here allow origins and I'm actually just going to paste those in because we haven't yet published the client we'll have to come back and um adjust this setting once we've published the React client and we're going to publish the React client to Versell the Versell platform which also offers us a free tier which is great allow origins and let's just I'm just going to paste these in as placeholder settings for now but we'll replace that once we've published the React code okay and that's our environment variables done and then it's just a case of deploying our web service so let's do that let's hit the deploy web service button hopefully everything works h required oh we haven't included that there okay we've got to include we've got to include dot slash app like that okay so that all looks right now okay let's do it wow okay deploy web service hopefully everything goes smoothly but when does it ever go smoothly cloning from repository and we're getting all of the status as it publishes our code we're getting all the relevant status and I'm really hoping this goes okay so you can see it's building building it uploading build now and it gives us a blowby-blow status of what it's doing regarding the deploy deployment process deploying build successful that's really great okay excellent deploying that's exciting and hopefully we don't have any issues warning unable to fund file oh okay that's just me i wrote unable to fundv file i meant unable to find file which is actually expected so that's not a fatal error so there's just a whole bunch of warnings here okay new primary port detected 8080 restarting deploy to update network configuration so you just note that you do need to to have your port set to port 8080 so it defaults to port 8080 so it's best to include that as your port port 8080 available at And there we have we've got it we've got our URL and I'm going to copy that URL because we're going to need to connect our client to that URL so I'm going to copy that URL to Notepad here so we got all our settings there and we're going to need to include that for our client side environment variable settings cuz it needs to connect obviously to the various endpoints and this is the base URL of our endpoints on render okay go back to render and that's looking good and let's see if we can test one of our end points through the browser just to see it says it's live and let's test the endpoint for slashmov like that we got to endpoint and it returns all the data for our movies which means we are ready to deploy our client on Versell excellent okay so now we're going to deploy the client to Versell now that we've got Magic Stream the serverside part of the Magic Stream application running on render we're going to deploy our client code to Versel so to do that we first go to the Versell platform the Versell website here okay and you can see I'm already um logged in here so you want to log in i've actually logged in with my GitHub credentials just makes it easier to deploy from GitHub okay and then once we've logged in to Versell you can create your deployment so first thing we need to do is go add new and then project like this okay and we want to import from magic stream movies so we go we import the magic stream movies repo so we hit import like that project name magic stream movies okay choose where you want to create your pro project and give it a name so project name Magic Stream Movies and you want to edit the root directory here so we want to edit this field here and we want to set this to magic stream client there continue so we've adjusted that route to where the our client code is within the repo the React code and it's detected that the framework is Vit for us okay and we can just call this project magic stream movies that's okay and then we want to add our we I think we've only got one environment variable so it's considerably less arduous than when we were configuring the server side environment variables so let's go to ourv file in our code and we just want to add this environment variable so vit API base URL let's include that there and then the value for that now is our render URL so let's go there and it's this here now that we want to include as our environment variable because that is the root path the base URL of where our API endpoints are okay so let's just double check that yep yep so that's correct so now we've we're not on local host anymore we're on we're on render so we include this URL where our magic stream server side code resides and this is the base URL for the relevant endpoints the magic stream endpoints okay so the output is dist run build so this is going to be our build command will be npm run build here directory is going to be dist so that's correct dist and I think that's it i think we're ready to deploy that now let's try it anyway let's go for it deploy deploy mint cued okay and off it goes it's doing its thing wow okay gavin's project you just deployed a new project sure that was pretty quick loading okay okay let's click on that to see if it's error fetching movies okay well that's not great um error fetching movies let's check what's going on there developer tools ah it's cause and that is because we need to configure this new URL within render so let's see what happens when we do that so we got to go back to render here and to our allow origins here we need to we need to adjust this setting here to allow the relevant origin so to do that we go edit and I'm just going to put a comma here and add the HTTPS address for where our React code is running let's take out that forward slash there and let's try again let's save rebuild okay it's rebuilding so it's redeploying our application again okay let's try again see if that makes it work okay so that's been deployed now just check um render let's go forward slashmov so that's working it's returning the expected data all of our movie data let's see what happens now when we try and run Magic Stream movies the client there we go working look at that so if I run one if I try to play a movie okay let's authenticate goth jenkins atotmail.com password one exclamation point let's see what happens it's logging us on hopefully it plays one of our movies and there it is look at that and our application is working perfectly what's on your arm O to a suspect baby oh my god let's go to recommended these are recommended movies for GT Jenkins let's play Unforgiven they're paying $1,000 to shoot them head on are you really going to kill them cowboys very cool and that's all working perfectly so let's log out and log in as Bob Jones so that we can include our own movie reviews bob Jones at hotmail.com password one exclamation point and we should be an administrator now let's go to recommended so these are Bob Jones's recommended movies and we can actually add reviews now so um let's create a review for The Hobbit okay the movie was awful i really hated it terrible what a an amazing movie movie i have never seen any like this okay submit review and let's see what sort of sentiment excellent and that's what you would have expected um so we got The Hobbit there is excellent within our recommended movies and this is fantastic everything's working i'm really happy with this got another Harry Potter there with the Empire Strikes Back let's see what Darth Vader's up to and where's Darth Vader a big turn in the strikes back singing let's There he is there's Darth Vader okay cool r2d2 there we go loads our movies i didn't put the spinner for the loading we need to I didn't update the spinner for that but that's okay we got the spinner going for the reviews here okay excellent movie was absolutely fantastic and everything seems to be deployed and working now so I mean that's amazing we've got our serverside code running on render we've got our client side code running on Versel and we've got our MongoDB database running on Atlas so it's fully distributed and everything's talking and all of our components are talking to each other and playing nicely together over the internet over HTTPS so all the data exchanged between the client and server are encrypted and therefore protected so nobody can steal our data and that is excellent i'm really really happy with that and let's log out excellent thank you for joining me on this journey i hope you've enjoyed going through this course as much as I enjoyed creating it so now you can create and deploy a fully secured distributed web application using Go React and MongoDB you've also learned how to integrate sophisticated AI functionality into your serverside web application code by prompting an LLM on Open AI through Langchain Go these are excellent skills to have as a developer moving forward into the future i hope to see you soon thank you and take care