neo4j is a powerful graph database that

enables organizations to model store and

analyze complex relationships between

data points its intuitive graph-based

structure offers a flexible and

efficient solution for diverse

applications making it a leading choice

for managing connected data this course

is taught by freecocamp team members

Farhan choudhary and Gavin long first

you will learn about neo4j and graph

database Management Systems as well as

how you can benefit from incorporating

them into your applications next you

will learn how to use a neo4j graph

database management system as the

back-end storage for a real world

application created with Java and spring

boot finally you will learn how to

create a front end with react to

interact with the data store with neo4j

neo4j provided a grant to make this

course possible so let's get started

hi and welcome I'm Gavin long so I

finally got my free code Camp t-shirt I

traveled all the way from the east coast

of South Africa to Amsterdam and met the

founder of free code Camp Quincy Larson

and he gave me my own free code Camp

t-shirt really great

so the point of the trip was to meet

Quincy and some of the core team members

of free code camp in person

it was about team building and

solidifying relationships between

co-workers the key Concept in this

course is in fact relationships

relationships between data the

relationships could be between people

between car parts between a T-shirt and

a human being

so relationships between homogeneous

data or lack data for example between

human beings or relationships between

heterogeneous data or different types of

data these relationships can be very

effectively represented within a graph

database management system

neo4j is a graph database management

system

in this part of the course I want to

present to you a brief overview

highlighting the significance of a graph

database management system before we

look at neo4j let's contextualize the

significance of a graph database

management system by looking at some of

the other popular types of database

Management Systems commonly used today

so let's look at three other commonly

used types of database Management

Systems before we discuss the graph

database management system

and why we would want to use a graph

database management system in our

applications

the most well-known used database

management system type is a relational

database management system examples of

relational database Management systems

are SQL server postgres and MySQL

in this type of database tables

represent entities for example a table

named employee contains rows of data

each row of data is divided into columns

or Fields containing employee

information for example the name field

contains the employee name the age field

contains the employees age Etc another

data entity could be implemented in a

table named Department each row in the

Department's table denotes an individual

department for example one row stores

information for the human resources

department at another row stores

information for the finance department

these data entities or database tables

within a relational database can be

explicitly related to one another within

the relational database management

system for example an entity may have a

one-to-one relationship with another

entity

so say employees and the cars they drive

a specific employee is related to a

specific car one employee to one car

so for example the employee entity in

this case has a one-to-one relationship

with the car entity

two database tables can have a

many-to-many relationship for example an

employee could be a member of multiple

teams and each team can contain multiple

employees so this means an employee can

be a member of more than one team and

each team can have many employees so in

this scenario the employee entity has a

many-to-many relationship with the team

entity

a database entity can have a one-to-many

relationship with another database

entity

for example the department table has a

one-to-many relationship with the

employee table meaning each department

can contain one or many employees and an

employee can only be a member of One

department at a given time

so this one-to-many relationship can be

established within the relational

database management system through an

appropriate schema design a database

schema defines how data is organized

within a relational database which

includes constraints such as table names

Fields data types and the relationships

between the relevant entities so for

example the one-to-many relationship

between the department table and the

employee table can be implemented by

including a primary key which is a

unique identifier for a row of data in

the department table and a foreign key

field within the employee table which

relates to the primary key value within

the department table the main advantages

of a relational database design is that

it helps facilitate data Integrity so

the data is not easily corrupted as well

as reduces data redundancy storage for

repeated values are reduced for example

if you stored Department data within the

employee table you can see here the

department short name field and long

name field values are repeated across

rows in the employee table whereas

separating the data into an employee

table and a department table means these

values are not repeated resulting in

Saving storage space as the data scales

the storage saving can be quite

significant

a technique known as normalization is

employed to design a relational database

I won't go into the details of

normalization here but please look up

normalization which is a key Concept in

relational database design so the

technique of normalization can increase

data Integrity minimizing data

corruption and also significantly

reducing data redundancy I.E

unnecessarily storing values repeatedly

the downside to relational databases is

the speed of retrieving data when

multiple joins or deep joints between

entities are necessary for example if

you want to retrieve analytical data

that summarizes data from multiple

entities a normalized database design

can cause the retrieval of such data to

be a slow process so way to speed up

data retrieval can be to use a

schemaless database management system

like for example mongodb this is a

document-based database management

system

the fact that the database design is not

bound by a schema like with a relational

database management system can result in

increased speed of data retrieval as

well as allow for better design

flexibility so you can see that each of

these database Management Systems have

their purpose and advantages

another type of database management

system may use a name value pair storage

structure like for example redis

this database management system is great

for caching frequently retrieved data

and therefore can help speed up a system

so we have briefly looked at three

common types of database Management

Systems relational database Management

Systems like for example SQL server or

postgres schema lists database

Management Systems like for example

mongodb

and name value pair database Management

Systems like redis so let's look at the

significance of a graph database

management system neo4j is a graph

database management system this type of

database management system is based on

the graph data structure all of these

social media platforms use graph

databases Twitter Facebook and Instagram

as we all know these platforms have

millions of users a relational database

management system would not be ideal as

the only storage facility for such data

because of the way the data is

structured would mean multiple joints

between entities would need to be made

when relevance data is retrieved which

would drastically slow down the data

retrieval process ultimately causing a

poor ux user experience because the

whole system is slowed down by these

joints that need to be made between data

so note that when designing a particular

system like a social media platform you

could incorporate both a graph database

management system and a relational

database management system into your

overall system design this type of

solution could combine the enforced data

Integrity inherent and relational

database management systems with the

complex relationships between data

entities or nodes that can be

facilitated using a graph database

management system for example the master

database could be a relational database

where data integrity and data redundancy

concerns are addressed by the inherent

benefits that we have discussed

regarding a relational database

management system

a graph database management system that

is appropriately synchronized with the

relevant relational database management

system can be leveraged to increase the

retrieval speed of data that for example

has complex relationships if data that

requires complex relationships was

retrieved directly from the relational

database management system the joins

required when retrieving the relevance

data could significantly slow down the

system and ultimately result in a very

poor ux user experience so by including

a graph database management system in

the overall system where it sits on top

of the relational database management

system as it were and is used to speed

up the retrieval of data with complex

relationships this could significantly

speed up the data retrieval process

resulting in a far better ux user

experience so this sort of system would

require that the data stored in the

relevant graph database management

system and the relevant relational

database management system be

synchronized so as to ensure that the

data is consistent across the two

different database Management Systems so

a system like this when designed

properly can leverage the benefits of

both types of database Management

systems for graph-based data structures

a database management system that

focuses on the facilitation of

relationships between entities or nodes

is essential and a graph database

management system provides a way to

store data based on the graph data

structure which is ideal for

applications that consume data where

entities have multiple relationships and

complex relationships with one another

think about social media platforms where

you have relationships between users and

posts likes friendships Etc and you have

millions of people using these platforms

a database management system is needed

that appropriately facilitates the

storage of data for a large number of

different types of relationships a graph

database management system allows for

relationships to be stored as data

within the graph database with a

relational database management system

the emphasis is on the entities and

generally the relationships between the

entities are not stored in the database

but rather General relationships are

established at the design level for

example you can see by the foreign key

field value that the employee is a

member of the HR department but there

are no details about the relationship

itself between the relevant department

and the employee saved in the database

this is something you could easily do

within a graph database management

system you could create a more detailed

relationship

you can easily save data regarding

individual relationships between

entities or nodes for example between an

employee node and a Department's node so

here is a whiteboard presentation that

shows how the relationships between

individual employees

and employees and departments could be

represented within a graph database

so here we see a node that represents a

department this node is appropriately

labeled Department

details related to this department node

can be saved to the database as

properties and related values so this

department contains a short name

property with a value of HR and a long

name property with a value of Human

Resources

here is our first employee node this

node is appropriately labeled employee

the employee node contains properties

and relevant values representing the

employee's name and age

here we see our first relationship this

relationship explicitly shows that the

employee whose name is Brad Jenkins is

part of the HR department

this relationship is appropriately

labeled member of

so you can see that this is a very human

readable representation of data the fact

that this employee is a member of the HR

department is clear

through appropriate labeling and

properties

so Brad Jenkins is a member of the HR

department

we are able to add properties to the

relationships themselves so the member

of relationship has a join date property

as well as an employee type property

you can see by this data when this

employee joined the Department and that

this employee has a permanent role and

is not for example a contractor here we

have the second employee node who has

the name Jenny Lane Jenny is also a

member of the HR department

we can save relationships between

employees within our graph database so

Jenny works with Brad we can also create

another works with relationship in the

opposite direction where Brad works with

Jenny

so you can see that we have a one

Department to many employees

relationship just like what was shown

earlier in the relational database but a

graph database is far more flexible in

terms of design when establishing

relationships between data points or

nodes not only can the entities contain

Associated data but the relationships

themselves can also contain data I.E

through properties and Associated values

so we could create even more complex

relationships between nodes for example

a chain of command hierarchical

structure related to our employee nodes

so here's an example of this where

employee James Stokes reports to another

employee Jenny Lane this is just a

simple example and of course this

hierarchical data structure could become

far more complex

so this presentation was only intended

to give you a basic idea of how much

more flexible the design of

relationships between entities can be

within a graph database and how complex

relationships between entities can be

easily implemented within a graph

database

I want to show you a brief demonstration

of using Cipher to query a graph

database and to do this I'll firstly

create a neo4j database on my local

machine I'll use my Mac platform to

demonstrate this I'll contrast using

Cipher and neo4j with using SQL in the

SQL Server database which of course is a

relational database management system

firstly what is cipher Cipher can be

likened to SQL or structured query

language Cipher is a text-based

declarative language used for querying a

neo4j database

what is meant by a declarative language

a declarative language is a high-level

language that describes what a

computation should perform these

languages must follow strict semantic

rules must be structured in a particular

way

in order to be valid and for desired

results to be returned from the

relevance database management system for

example using SQL in order to bring back

all employee records from the employee

table you would appropriately execute

the following SQL code

the SQL statement simply returns all

rows and all columns from the employee

table

using Cipher if you wanted to return all

employee data you would use a query like

this this query would return all

employee nodes and relevant

relationships although When comparing

these two query languages they are of

course semantically different but

principally they are the same and that

the relevant declarative text-based code

is paused by the appropriate database

management system converted into

appropriate instructions that are

executed under the hood as it were and

the relevant results are then returned

from the relevant database I'd like to

demonstrate using Cipher within a neo4j

database I'll contrast executing Cipher

against the neo4j database with

executing similar queries against a

similar database implemented in a SQL

Server database you don't need to follow

along with this part of the course but

of course you can if you want to so if

you want to follow along you can install

neo4j on your Mac or Windows platforms

by following the instructions at this

URL I have already installed neo4j on my

Mac platform

if you want to install a free version of

SQL Server please navigate to this URL

but as discussed this part of the course

is only intended to be an overview and

not a detailed explanation so it's not

necessary to follow along with this part

of the course I've created these scripts

on GitHub for this part of the

demonstration so I'm going to copy this

script from GitHub and run it in neo4j

to generate My neo4j Graph database so

with the neo4j desktop let's create a

new project I'm going to call this

project sample project

then let's create a new database I'm

going to name my neo4j database company

DB

let's give our database a password then

let's press the create button

and now we need to start the database so

let's press the start button to do this

then to execute our Cipher queries let's

click this open button and open the

appropriate window from which we can

execute our Cipher queries

so the first thing we need to do is copy

the code for creating our graph database

so let's navigate to this appropriate

GitHub URL let's copy this code from

GitHub and appropriately paste the code

within the window that we can use to run

our queries through neo4j desktop

great and let's click the play button to

create our graph database excellent and

we are presented with appropriate status

information telling us that our graph

database has been created successfully

I'm going to run this script within SQL

Server to generate a similar database

the difference of course being that this

database is designed as a relational

database

so let's appropriately copy this SQL

code from GitHub

let's launch SQL Server management

Studio

let's run our SQL code to create our

database I've already created a database

named the company DB test

so I'm going to create a new database

named the company DB test 2.

then I need to change the relevant SQL

script to point to the new database that

I've just created let's press the play

button to run the script and create our

database tables and relevant data

great

so I'm going to run a few queries so

that you can get a basic understanding

of the differences and similarities

regarding Cipher and SQL so let's run

this query against the neo4j database

great this query brings back all the

employee nodes and relevant

relationships

in order to retrieve all employee data

from within the relevant SQL Server

database you could run a SQL query like

this

and let's say we wanted to retrieve

particular data about a particular

employee

so you could run this Cipher code like

this

within the SQL Server database you could

run a SQL query like this

okay so now the next query I'd like to

run is a little bit more interesting so

in both these two databases we have

expressed a hierarchical relationship

between employees in the neo4j database

this is implemented through the report 2

relationship established between

employees in the SQL Server database the

same hierarchical relationship between

employees is expressed but through the

use of what's known as a self-joining

table

so in this case the employee table joins

to itself through the relationship

between the report to ID field and the

ID field which stores each employee's ID

so in the employee table the

hierarchical relationship between

employees is established through the use

of the reports to ID field which stores

the employee ID of the relevant

employees immediate boss if you like so

employee Brad Jenkins is at the bottom

of the hierarchy as it were so if you

wanted to query the SQL database to get

the chain of command from Brad Jenkins

to the top of the hierarchy you need to

run a query like this

as you can see this query is really

difficult to understand at first glance

and recursion is used to get the desired

results

a graph database is far easier to

understand in terms of retrieving such

tree data structures from the relevance

database

so to use Cipher code to retrieve the

relevant hierarchical data starting with

an employee at the bottom of the

hierarchy Brad Jenkins you would run

Cipher code like this against this

particular database

great okay so I hope this part of the

course has given you at least a basic

understanding of a graph database

management system and how you can

benefit from incorporating a graph

database management system into your

applications Farhan will now go into

more detail where he'll use a neo4j

graph database management system as the

back-end storage facility for the

server-side part of a real-world

web-based application I will then follow

Farhan and create the front-end code for

this application using react

hello everyone welcome to this course I

am Farhan hassin showed you one of the

instructors and I will walk you through

the entire process of working with a new

4G database as well as the development

of an API powered by Spring boot

framework

now at this point of the course I will

show you how you can create a new

account for the neo4j aura Cloud

database service

how to create a new database and run

some simple queries

now keep in mind that this is not going

to be an exhaustive list of all the

commands and queries that you can run

inside in your 4J database what I am

going to do is I will show you how you

can create a node how you can create

relationships how you can delete a node

update a node and some other basic crowd

operations in a word I will teach you

whatever is necessary to create the API

now to begin with we will navigate to

neo4js website I will leave the link in

the description so you can follow along

and once you are here you will use the

get started free button to create a new

account now start free and if you wish

you can sign up for a new account using

your email address but what I am going

to do is I will just use my Google

account to log in now I have

successfully logged into neo4j Aura and

I hope you have done that too so this is

the dashboard that they give us you can

see there are the instances that we are

going to create Now to create a new

instance of a database just click on the

new instance button and it will give you

a free templates like if you are a

beginner you can start here and it will

give you some data to start with what we

are going to pick is an empty instance

because I want to show you how you can

create new nodes relationships labels

and things like that

so we will hit create then they will

generate a password for us now you can

copy that password using this button

or even better you can use this download

button to download all the credentials

as a text format now I am not going to

do that yet but I will show you later on

how that works out

so once you have copied the credentials

click on confirm I have copied or

downloaded the above credentials so that

they will know that you have the

treatment Shields because you you cannot

get it once you have gone past this step

now you need to continue

now the creation process of a new

instance is a bit long it can take

around a minute or so so what I'm going

to do is I'm going to pause the

recording and come back once the

instance has been created

so it looks like that the database has

been created uh it's it's running on gcp

or Google Cloud platform and since I am

from Asia I am from Bangladesh

the region closest to me was Singapore

now we also have a connection URI now to

connect to this instance all you have to

do is click on the open button but

before that just in case you want to

rename your instance you can go here

and you can use rename I am not going to

do that though so I will just open it

and they will ask for a password so I

will just paste in my password

and the database user will be neo4j

you don't have to change any of these

things just click connect now once you

have landed on your workspace you will

see that there are no notes and

relationships whatsoever

but I would like to show you how you can

create new nodes how you can create new

relationships between them how you can

update a node now the cipher query

language is really expressive and this

is really easy to learn if you have

experience with some other query

languages like SQL but even if you do

not have any experience with that Cipher

is not that hard to learn the cipher

documentation is an excellent piece of

resource they also have a cheat sheet

now I will leave a link to this cheat

sheet in the description of this video

you can visit these and you can

essentially learn about how to read all

the notes in a database using the match

clause

or how to filter them using where return

variables

and under write queries you can also

learn about creating new nodes levels

and relationships

set for updating them delete and remove

for getting rid of notes and labels and

things like that I would suggest that

you bookmark this Cipher cheat sheet in

your browser so that you can refer back

to it whenever you need and also the

documentation now I will go back to our

workspace and I will start creating a

bunch of node and I will explain how you

can do that as well

so let's switch back to our workspace

and the first thing that we want to do

is we want to create a new node so we

will begin with the create class and a

set of parentheses

then we will put a colon here and we

will write the label of the node so a

node is an independent or a single

entity inside your database and it has

to be labeled something

for example if we create a node in our

database with the name Farhan has in

children it would be hard to say what

this node represents does it represent a

user does it represent someone's pet cat

what is it

so we are labeling our node with the

label user

and then we will put some properties

here the first thing I want to put is a

name for this user so Farhan

Name colon

Farhan hasin Chaudhary then I will put a

comma the next is a username this is a

unique username and we will use it to

uniquely identify every single user so

we will say username is just farha and

that's all I am going to put here now I

will execute the command using the play

button

and looks like we have a new label here

and if I click on this it will return

everything under this label to us now

instead of clicking on this label we can

also use the match class so we will say

match

and we want to match for all the nodes

in our database we use a return

all the nodes now this is just a

variable by the way if you want to call

it nodes

nodes you can do that as well just make

sure you are not using any of the

reserved keywords for example if I

remove the Acer you will see it becomes

red which means it's a keyword like

match and return we cannot use it so I

will change it back to notes I will run

the query and as you can see we have all

our nodes here

now let's go ahead and create two more

users so I will say create

and this time I want to actually create

multiple users with a single create call

so I will keep pressing shift and press

enter to come on the next line

I will start a set of parentheses I will

create a new user with the name

John

and his username

will be

just John

then outside of the last parenthesis I

will put a comma

and I will again keep pressing shift and

press enter to come to the second line

and I will open another set of

parentheses I will create another user

and her name

will be chained

and username will be just J

it has to be stream

now if I end this query with a semicolon

at the end I can just press on this play

button

and as you can see I have successfully

created two new nodes

at four sets of properties so if I say

match

in for notes and return in

I can get all the nodes in our database

so I have foreign I have John and I have

Jane now the thing is all these three

nodes are of the user level

so it would be nice to have another

label on our system so let's go ahead

and create a bunch of courses

Now to create a course we will use the

same

syntax as before and you can use your up

Arrow to bring back your old queries

I will just make some changes to this

query I will switch user with course

in both cases

I will replace title

we

C plus plus course

and the username with identity fire

this is an unique identifier for the

course which is CPP and the next course

we will have is title

Java Script

course and we can call it Js

finally we can have another course

called

C sharp course

and

its identifier

will be C sharp

whether I have made a mistake here it

should be identifier I'm not username so

let's look at our query we are creating

three courses with title C plus plus

course Java Script codes and C sharp

ports they all have uh unique identifier

cpp.js and c-sharp so I execute this

command

uh looks like I have a mistake which is

I didn't put a comma at the end and how

did I know that so it as you can see

neo4j is actually telling me that there

is some problem in the beginning of the

course title C sharp line but I can see

that it's totally fine so it's highly

likely that the mistake is actually at

the end of the previous line so that's

how I thought that maybe I have missed a

comma so I will run this query again and

I have created three new nodes and I

will say match

in and I return

in

so as you can see I have all my notes

Here There is Jane John and Farhan has

seen jewelry these are the users and C

plus plus JavaScript and C sharp these

are my courses now it would be nice if I

could create a relationship between Jane

and this C plus plus course John and

JavaScript and foreign C sharp course so

I want to essentially enroll them in

this course

but before that I would like to show you

a few variation of the match Clause so

right now when we say match in return in

it actually returns all the nodes in our

database

what if I want to only get the course

notes or the user nodes we can do that

by

defining a label after this variable

like this so I want all the users

return in as users you can actually do

that if you use as

you can actually rename your variables

to make them more user friendly so I am

matching all the node of label user and

I returning them as users I'll press

play

as you can see I am now getting only the

users and not the courses

likewise I can replace n with C or let's

just keep it n

since that's just a variable

I am picking the course as instead of

users I will say courses

by the way you can press Ctrl enter to

execute your query without pressing the

play button so right now I am getting

only the courses

but what if I only want to get one user

for example I only want to match for the

user with username Farhan

so I can do that I can same match

you for user so I input user then I will

use the where plus this is used for

filtering or query results so I would

say fire U dot username

is equals Farhan

return

U as

far

so what I am doing essentially here is I

am matching for a user type

that has the username Farhan I am

putting it inside you and I am returning

U as farha let's run this query

and as you can see I am getting Farhan

has in two three back

now let's add a course to the mix so

after I am saying U colon user I can put

a comma start another set of parentheses

I can say

C

course

so where U dot username equals Farhan

and

C dot identifier is equals CPP

so return U comma C

but this time I am removing the as

pre-word

and what I'm doing here I am matching

for a user and a course

fire the username is equals Farhan and

course dot identifier is equals CPP

finally I am returning U and C let's see

if it works

as you can see we have successfully

found the C plus plus course and the

Farhan has into three users now we are

actually ready to create a relationship

between these two nodes

so I will bring back my old query

and instead of returning them

I want to say create

because I want to create a new

relationship and I will first take the

user

so here I have the user

then I will put a dash

then a set of square braces and inside I

will say colon

enrolled in this is the type of

relationship this user is going to have

with the course

so what we are doing we are matching for

a user and a course where the username

is Farhan and course identifier is CPP

then we are creating a relationship what

kind of relationship you say we are

creating an outgoing relationship from

the user and how do I know that as you

can see there is an arrow going out from

the user towards the course so this is

an outgoing relationship from the user

and what is the type of this

relationship the type is actually

enrolled in

now we can actually save this

relationship inside a variable called r

and I can say return

r

so if I press the play button again

yes looks like I have successfully

created a relationship so the

relationship actually starts at Node 1

and ends at node 4 which means that the

ID of Farhan hasin choudhary is 1 and ID

of CPP is four if I go to user

you can see for Han hassin children has

the id1 and if I go to course

C plus plus course has the ID of four

okay and the type of relationship is

enrolled in there are no properties

whatsoever there are the element ID

start note element idea things like that

we do not have to worry about them

so that's how you can create a

relationship between

these two nodes

so that's how you can create

relationships inside of neo4j now let's

see how you can update a property of a

node so if I want to update the name of

Jane to Jane Doe and John to John Doe I

can do that

first we have to find John in our

database so we will say match

a variable U of type user

where

U dot username

equals John

so now we are matching for a user with

the username John

then we want to use set this is another

clause in neo4j Cipher language so it's

a set

U dot name

equals

John Doe

and if I execute this query

looks like it has done its job so if I

go back and instead of setting it if I

return

you

John has now become John Doe

we can do the same thing with Jane

because why not they are related

Jane and we can say

okay so I want to return J

return U

there you have it so now you have

learned how to match nodes in the

database you have learned how to create

new relationships update values of an

existing node now that's pretty much all

you are going to need for the API

development process but I would like to

show you one last Clause from The Cypher

query language and that is the exist

clause

for example if you ever want to check if

Farhan hasin tutori is enrolled in the C

plus plus course or not you can do that

using the exists Clause so we will begin

with match and what do we want to match

we want to match for an user

and we want to match for a course

where

U dot username sequels Farhan

and

C dot identifier is equals CPP

now we have an user with the username

Farhan and we also have a course with

identifier CPP next what we want to do

is we want to say

return

and we will call the exists function

this is a function and that's why we

have these two parentheses and inside

this parentheses we want to describe a

relationship

so the relationship we are looking for

or checking for is

whether user

is

enrolled in

the course or not

okay so you will finish our query what

we are doing we are matching for a user

with the username Farhan and we are

matching for a course with the

identifier CPP then you are checking

whether a relationship between them of

type enrolled in exists or not

so I'll press the play button

and it says true

now if I switch CPP for something else

like JS we have the JavaScript course

and I hit the play button

you will see I am getting false

now to be honest that's all you need to

know to go through this entire course

but Cipher is really excellent query

language and if you have plans to learn

a lot more neo4j than I have showed you

in this course I would suggest that you

go through the documentation you go

through their

cheat sheet and you practice making your

own databases creating them in a lot of

different scenarios and learn as much as

you can about this database system

now I am going to finish this lesson

here but in the next lesson you will see

me using an initializer Cipher script to

create a new database for our API

and from there we will begin working on

our Java code

in this part of the course we are

finally going to start working on our

spring boot API but before that we need

to set a few things up

first thing first go back to your neo4j

aura database dashboard and delete any

old instances that you may have created

while working with the API we want to

start with a clean slate so that we can

be sure of the fact that some old data

will not mess up our API to delete an

old instance you can use the delete

button here

and then you will have to write the name

of the instance

it's just to make sure that you are in

your right senses and you are not

sleeping or anything

click on the destroy button

and wait until the instance is being

destroyed

Now to create a new instance

use the new instance button

choose empty instance and click on the

create button

make sure you download the credentials

file because we are going to need it

later on

and also let them know that you have

downloaded the credential file by

clicking on this checkbox click on

continue

now creating a new instance may take a

few minutes so what I'm going to do is

I'm just going to speed up the process

in this video so that you don't have to

wait around for long

once the instance has been created click

the open Button

accept their terms and conditions

and when they ask for the password open

up the credentials file you just

downloaded

just highlight the password Here

copy it

and paste it on the empty password field

click on connect

and we have successfully connected to

our new instance

now as you can see this instance is

completely empty but we will need some

data to work on our API

for that I have already created an

initializer script that you can use to

initialize the database

for that

go to the code repository on GitHub I

will leave the link on the descriptions

below

then go inside the data folder

you will find a file called

initializer.cipher

open up this file

and go to Raw

highlight all the lines and copy with

Ctrl C

then go back to your aura DB dashboard

paste the code that you just copied

and then press the play button to

execute the code

as you can see the code has successfully

created some notes relationships and

property keys

now the first note that I would like to

show you is the user node

now this represents all the users in our

database

for example if we open up John Price you

can see that it has an ID

it has a name John price it has an

encrypted password hash

it has roles and it has username

now the significance of each of these

properties will become more clear as you

start working on the API but for now

just keep in mind that

all these properties are important for

our application

next up we have course

a this represent

each of the courses that will be

available on our system so if I click on

the Blazer shopping cart course you can

see that it has an ID of pen

it has a string identifier that we are

going to use to uniquely identify this

course

it has the name of the teacher and a

title now you may argue that we can use

the ID to uniquely identify this course

while that's correct

I prefer using some other unique string

or identifier to identify my course

or entities I just do not like to rely

on the auto generated ID fields in any

database at all

finally we have the lessons these are

the individual lessons that belongs to

some course

for example

if we click on the Blazer

webassembly and web API on dotnet 6.

listen you can see it has an identifier

just like the courses it has a title and

it also has an ID

now these lessons here has a belongs to

relationship with the courses which

means each of these lessons belongs to a

certain course

for example if I zoom in on the Blazer

course

as you can see

all these four lessons belongs to the

Blazer shopping cart course

finally there is the enrolled in

relationship

which is between the individual users

and courses

so for example

John W here is enrolled in the Blazer

shopping cart course

now that we are done with our database

we can finally start working with our

API

for that you will need jdk pre-installed

on your computer

now you can download the jdk from

oracle's website I am going to use jdk

17 since that's the latest long term

support release

but you can also go with jdk 20 it's

totally up to you

one more thing for those who are

watching this course on Linux it's

totally fine to use any of the open jdk

builds for example

I myself am using the Microsoft build of

open jdk instead of oracle's official

build

and given the complexity of our API it's

not going to make

any

change or it it will not break anything

whatsoever so feel free to use uh any

open jdk build that you like you will

also need a good ide to make your you

know programming process more fun I'm

going to use IntelliJ IDEA ultimate here

I have a student license so I'm going to

use that but if you want you can also

use the Community Edition to be honest

if you look at the feature comparison

here most of the essentials are

available under the Community Edition we

do not need the HTTP client thing since

we are going to use a third party client

anyway we do not need database tools or

SQL or any of these things for this

project at least

finally

we need to initialize our project

for that we are of course going to use

the spring initializer

now I will leave the link to Spring

initializer down in the description so

once you have navigated to this page we

will make some choices here and we will

generate our project boilerplate

first make sure that you are picking

Maven under project here you can also

use griddle if you have previous

experience with the tool but I am just

going to pick maven the language will

remain Java for the spring boot version

I am going to use 3.1.0 since that's the

latest at the time of recording but if

some other version comes out later on

while you are watching this course feel

free to use that just do not pick any of

the snapshots you can stable release

now under group you will have to type

or you should type some domain in

reverse order for example my own domain

is farhan.dev

so I will just type in Dev Dot

then artifact is just a project name so

I am going to call it spring

boot

neo4j you may call it anything that you

like I will also put a sensible

description like a simple

API powered by

you know

or Che

finally make sure that you are picking

the correct Java version from down here

I am going to pick 17 since that's what

I have installed on my computer but if

you have installed 20 make sure that

you're switching to 20. for packaging

format we will go with jar since that a

lot widely used than wire format so

let's not change that

another most important part adding

dependencies

now this is needless to say that we will

need a number of different dependencies

for our project but

for starters I will just pick spring web

and

the devtools

now you may think that

Why not pick the new 4G dependencies

while we are here well I could have

picked them but one thing that I have

noticed in the past

is that many people face difficulties

while they try to add new dependencies

to a project that they have already

initialized

so I thought why not to start the

project with just the bare bones uh you

know essential dependencies only and

then as we go forward we will keep

adding new dependencies to our project

that way it will represent a much more

realistic you know development workflow

and it will also teach you how you can

search for new dependencies and add them

on the flag

so take a good look at all the choices

you have made and once you are happy

with them click on generate

it will download a little zip file to

your computer

that contains all the project 5.

I'm going to just open up the zip file

using

no Windows Explorer or WinRAR or

whatever you like

copy the project folder

go to a place that where you usually put

your codes in I usually put them in

individual directories so I'm going to

paste it here

and once I have done that I will just

open up my IDE which is

IntelliJ IDEA

I will open up my project

input neo4j okay yes we trust the

project

and once you have opened the project for

first time I would say give it some time

to you know cash all your dependencies

uh read the source files and things like

that so the ID knows about your project

and it can help you along the way

hello everyone

I am your instructor Farhan but I am

from future

now in the next lessons you may see a

little problem with the video in some

cases when I am trying to create a new

class or maybe a new package you may see

that this menu is not showing up now

this is because my recorder was only

recording one window in my computer so

all this sub menus were invisible

so whenever you see me creating a new

package all you have to do is right

click

on your main package go to new and then

package likewise when I am cheating in

new class you have to pick Java class

from here and you can switch between

interfaces and classes right from this

menu

another thing that didn't work in my

previous recording was the code

generation thing

so whenever you are creating a new model

you can use the alt insert key

combination sorry inside the class of

course

let's get rid of the Constructor and you

can use alt insert and you can then

generate a Constructor you can generate

Getters and Setters and equals and hash

code two strings and all those

boilerplate methods so what I did in

those videos is I picked Constructor and

then I just ignored all these properties

since you are trying to create an empty

Constructor and then press ok

so

a good chunk of the course if you see

all these sub menus are invisible just

don't be alarmed it was some problem

with my recorder everything should work

as is in your computer

now back to the past

as you can see IntelliJ IDEA has finally

finished resolving all the dependencies

in my project so I am finally ready to

write some code now before we start

writing Java code I'd like to address

something really important I hope that

you remember back when we are generating

our project we didn't add all the

dependencies to our project because I

wanted to show you how you can manually

add dependencies later on

so this is the time we do that first

open up your pom.xml and find this

dependencies part

inside that

you can see that there are three

dependencies already described

you may recognize the spring boot

starter web and springboard Dev tools

dependencies from a few moments ago when

we added them to our project but there

is also another thing called Spring boot

a startup test now this is for testing

our code and in this course we are not

going to talk about that

now the first dependency that we need to

add to our project is

called

the spring

put a starter

data neo4j and the group ID will be org

Dot springframework.put

now IntelliJ IDEA is being really

generous here and helping us out with

writing the dependencies but I suggest

that you make sure that you are typing

them out correctly because if you make a

typo or write wrong name then you may

end up not getting a dependency or even

worse getting the wrong dependency

now the second dependency

that we need is called the neo4j Java

driver and the group is org.neo4j dot

driver

now whenever you add or remove a

dependency from your palm.xml file you

have to right click on the palm.xml file

find Maven and reload project

depending on how many dependencies you

have added or removed from your project

the resolving process may take a while

but in my case it has finished in a

Flash

now I will close the pom.xml file and

start creating the packages that we will

need

now inside our main package

dave.farhand.springboot neo4j it may be

something else in your case but

inside our main package we will create

four new packages now the first one

will be called

config

then we will create

another package called models

then we will have

the repositories

then we will have

the services

and our controllers

now these are not the only packages that

we are going to need we will also need

some packages to hold our query results

our data transfer objects our requests

and things like that but we'll work on

them later for now these are the ones

that we need now once you have all the

package ready to go let's create the

model for our first node which in this

case will be the course so let's create

a new Java class

and call it

course

then inside we will Define all the

necessary properties of this class so

the first one is an ID and it has to be

of type long

then we will have private string

identifier which will used uniquely

identify our courses

then private string title

and private extreme teacher

then we will create a Constructor now

you may write the Constructor by hand

but inside IntelliJ you can just press

on alt and then insert to bring up this

nice generate menu

from here you can select Constructor

and for our Constructor we will not

select any of the properties

now for

spring data neo4j this is one of the

requirements that you create an empty

Constructor inside your model classes so

this is what we are going to do

then we will

say alt insert and we will select

Getters and setters

highlight identifier title and teacher

we will not highlight ID

and why you will understand in a bit

click OK and now we have all the Getters

and Setters for our

private properties

now creating a class like this will not

suffice who you will also have to

annotate this class with the node

annotation

This Way Spring boot will know that this

class is actually a new 4G node and has

to be managed by the dependency

injection system

then inside the class we will also

annotate the ID with two separate

annotations the first one has to be the

ID annotation

to let springboot know that this is the

ID for the node

and then we will say generated value

which means we want this value to be

automatically generated and we do not

want to write it by hand

now since this ID is an automatically

generated value we do not need a set of

file and since we are not going to use

these two uniquely identify our course

we are going to use this string

identifier instead we can just get rid

of the getter as well

so now that we have a course class the

next

the next thing to create is a repository

so let's go inside the repositories

package go to new Java class and we will

call this course

Repository

and it has to be an interface by the way

once you have created the interface it

is going to extend

the neo4j repository interface which

comes with our dependency

and this is a generic and inside

we will write the name of the class that

we are working with which is course

and we will also write the data type of

the ID of this course so in our case it

will be learn

now this way of creating a new

repository is nothing unique

to neo4j actually if you have experience

with spring boot in the past and you

have worked with databases like mongodb

or MySQL you may have created interfaces

like this

now let's close our repository and model

and let's go inside our service

we will create another Java class called

course service

and we will have to annotate this class

with the service annotation

then inside this class we will need a

reference to our course repository so

you will say

private final course repository course

repository you can name it anything

and then we will generate a new

Constructor where we will take the

repository now that's it now since we

have annotated this service class

with the service annotation and also the

repository

extends the base neo4j repository did

this will be on automatically injected

into our class

now inside here we will create another

public method we will say public

and it is going to return a list of

courses

get all courses you can name it anything

by the way

then we'll say return course repository

dot find all

that's it

finally we will create a controller by

the way if you think that I'm going to

fast just be patient once I have created

the model repository service and

controller for the course I will explain

the entire workflow or the architecture

of the application what I am planning to

do and what you should do in any of your

spring boot applications

so finally I will create a controller

new Java class and it will be called

course

controller

now I need to annotate this class with

the rest controller annotation

so a springboot knows that this is

actually a rest controller and this is

going to describe a few endpoints

then I will say request mapping this

another annotation and I will describe

the general endpoint

so I'll say slash API slash V1 slash

purchases

slept

now inside this public class I will

create a

private reference to our service class

so I will surprise it

final

course service

or service

then I will generate my Constructor

and I will create my first method which

will be public

and it will return our response entity

yes response entity of type list

course

I'll explain what it is in a bit just

just hold on for a little longer

and I will call My Method course index

you can call it anything this is just

going to list out all the courses in our

application

and then we will say

return

new response entity

and inside here I will call

courseservice Dot get all courses then

comma

HTTP status dot ok

and with a semicolon then I will also

have to annotate this class with a git

mapping call

and this

okay let's let's take a final look at

our application and make sure that the

code is working fine yeah that that

looks okay

now

we have a model

that describes the course node in our

database

and as I have said a few times already

in our database the courses have an ID

which is an auto generated field

we have a string identifier a title and

teacher then we will need

an empty Constructor for all our models

then we have some Getters and Setters to

work with this model we create a

repository interface which in turn

extends the new 4J repository interface

now what is this interface actually to

see what this contains we can press on

the control button and then click on

this Repository

once you are inside the source code you

can see it gives us a few useful methods

and one of them is find all so whenever

I am going to call find all in this

repository it will actually return all

the courses in our database in a list

format as you can see it says list t

there is also find all by ID there is

also find all with assort functionality

so we can sort the courses so this is

what a repository holds there is also a

safe all method but let's ignore that

for now

now since we have a repository we can

used it inside a service you do not want

to use your repositories inside a

controller directly because that's not a

good thing to do you should always

create a service layer that talks to the

repositories and use that service inside

your controller layer always remember

that controller should be responsible

for taking and giving back data they

should not

worry themselves about the underlying

architecture of your database or your

models or anything like that

so in this case we have a service class

that makes a reference to our course

Repository

and we have a single method that gets

all the courses for us

one thing that I'd like to mention here

that you may have seen the auto wired

annotation that comes with the spring

Boot and maybe instead of creating

a Constructor like this I could have

just annotated this field with autoware

but in some cases that's not the best

thing to do now since we have our

service we will go to our controller

that describes a single endpoint which

is API V1 courses

inside this method we are just returning

a new response entity this is one of the

classes that comes with spring Boot and

you can use it to return a response to

the user

and this response entity thing takes two

things

one is body and the other is the status

code now for body we are giving it the

least of the course since it's a

response entity of type list course

and for status code maybe we have

written 200 by by hand but instead of

using some magic number what we are

doing here we are using the value of ok

inside HTTP status this is actually an

enum this enum here actually describes a

lot of other status codes as you can see

it has continued it has switching

protocol processing checkpoint okay

created accepted we are going to use

created very soon no content and a lot

of other things so we are going to make

use of this enumerator a lot in our

application

so

technically our application is ready to

run and give us the list of courses but

we have to make one final adjustment and

that is we have to

let the application know about our

database that we have created earlier

for that we will need another dependency

now the dependency that we need is

called Spring dot EnV

so we'll say

dependency and inside artifact ID we

will write spring

Dot N

and inside group ID

we will say

n e dot all

words s c h w a r c

okay and we will also describe a version

for this dependency

subversion

3.0.2

we'll save the form.xml file and use

this little button here to load the

maven changes let's click that and wait

until the dependencies are being loaded

so as you can see it has turned white

from Red which means the dependencies

have been loaded successfully

next up we will create a very simple EnV

file

inside

SRC main resources

and we will just right click on the

resources folder new and file we will

call it dot EnV now this EnV file will

contain three values the first one is

or uh

URI

the second one is or

user and the third one is aura

password

now as you may have already guessed the

aura URI is a URI to our neo4j server on

the cloud the aura user is the name of

the user or the username and finally the

password is for the password for the

database now the order user is going to

be always new for J

and I will just quickly copy

the URI and password from my new

fourject credentials file

okay so I have copied my URI and

password from my credentials file and

these are going to be different in your

case since your instance is different

from mine

now once you have created this dot EnV

file next up you will need to open the

application.properties file now inside

we are going to reference the three EnV

values we that we have so I will just

say spring Dot

Neo

4J

dot URI

and then

a dollar sign

curly braces and

aura

URI

next up is spring dot Newport

dot Authentication

dot username

again another dollar sign

early braces and aura

user

finally it says spring dot neo4j dot

authentication

dot password it's a dollar sign credit

braces or a password

now why did we do this why we didn't

just put all these values inside our

application.properties class

now the thing is these values that URI

to your instance the user password and

the username are secret values these are

things that you should never put on your

GitHub repository

since the application.properties file is

a part of your application we will not

put our secrets directly into this file

instead we are going to create an EnV

file

and these three variables will be loaded

into our local environment

and then the application properties will

be able to find them using this dollar

sign and curly braces Index this is how

you actually access any of the

environment variables inside your

application

now I would suggest that you also make a

copy of this file

and call it.anv Dot example

then get rid of these values

save the example file

and make sure that your dot EnV file is

being ignored inside your git ignore

file so inside here I will say

dot EnV

and then set dot EnV so this will make

sure that the dot EnV file is not added

to your GitHub repository

instead you will have an example file

that you can use a reference to create a

new enp file

okay so since we have our model

controller suppositories and services in

place

and we also have our EnV file created

our application should be ready to

connect to our database and give us a

list of the courses let's test it out to

do so you will have to click on the play

button

so far it looks like that the

application is running fine

so what we are going to do is we are

going to switch to an HTTP client called

insomnia and we will test out our

application by sending a request to it

now inside insomnia the first thing that

you should do is create a new folder

because we want to keep our requests

organized now I will call it spring

boot neo4j

and I will move or just just let's get

rid of this request now inside this

folder I will create a new HTTP request

this is going to be a get request of

course

and this is going to

http

localhost

8080 slash API slash V1 slash courses I

hope that you remember

and let's also rename this to

get all courses or something like that

okay

it really doesn't matter to Grants

now let's send a request

yeah and looks like we have our list of

courses as you can see we had four

courses in our database and all of those

are being taught by Gavin long and the

list contains all of 10 and we also have

a 200 okay status quo I hope you

remembered that we had added that little

status HTTP status dot OK at the end so

that's why you are getting the nice 200

okay response here so congratulations on

creating your first API endpoint in this

application

next up we are going to work on the

lessons and we will implement the

relationship between the lessons and the

courses and we'll also make sure that we

can

see the details of this individual

courses along with the list of the

lessons

so I am back at IntelliJ IDEA once again

and now we are going to start working on

another endpoint for our application

so let's go back to controllers now the

next endpoint that I want to work on is

an endpoint that's going to return the

details of a single course

so if I go back to insomnia let's make a

copy of this request

okay

single course

and we will just pick an identifier from

all courses request

we will put it inside the single request

and we want to make something like this

now I actually like to start working

from the lowest level of our application

which in our case are the repositories

so I will go back to course Repository

and inside we are finally going to

create a custom method

now this custom method will return an

optional course

and we will call this method find

course by

identifier

and then

we'll take a string identifier

now the nice thing about these neo4j

repositories is the fact that they are

really smart

so just by the name of this method find

course by identifier

spring boot will know that we are trying

to find a course by using identifier

property

and we will also need to pass the

identifier by string now if I remove

this part and I write it again I will

say find

and as you can see IntelliJ IDEA

actually recommends me that

find course buy so you can sense that

maybe you are trying to find a course by

some parameter so I'm going to pick find

course by and then now it started to

suggest me the possible property names

and from this list I am going to pick

identifier

and inside of course we are going to

pass the identifier let's say string

identifier

okay now we have the repository method

we will go back to our service

which is the one step higher level of a

repository and we will say

public course

and we will call it get

course

by identifier

and we will take

the identifier as and string

now inside the method we will say return

course repository Dot

find course by identifier identifier

now I hope that you remember that the

method actually returns an optional type

and we are trying to return a course

object from this method so what we are

going to do

in the second line we will say

R else true so if you do not find this

course or if you do not find a course by

this identifier

true

site there will create a new Lambda and

we will send new

response

status exception

and we'll say HTTP status quote

Dot

value of

404

so what we are saying is first use the

find course by identifier method inside

the course repository and try to find a

course by this identifier the identifier

that I have passed

and in case you do not find a course

using this identifier please throw a 404

not found response exception status

that's what we are doing

let's close this service

and inside our controller

we will create a new get mapping

public

spawns entity

of type course

and we can call it

course details

it has to be a get mapping

and it will

give us an identifier so we are will get

the identifier from here

and inside course details I will set

path variable

which means we are going to get a

variable from the path

and it will be string

identifier

yeah that's it you can call it anything

else if you want

now inside the method what we are going

to do is we will say return or or let's

just create a course

equals

or service dot get course by identifier

we will pass our identifier right here

and then we'll return

a new response entity

and we are going to say the course is

the body and HTTP status dot okay

okay yeah

so that's it what we are doing here is

we are getting an identifier in our path

we are passing that identifier to our

service layer that service layer in turn

will call our repository and get us the

course that we are looking for

now essentially this should work so

let's try out let's rerun our

application

go back to insomnia and let's execute

the git method that we created a bit

earlier that's it send

so as you can see it actually Returns

the course that we are looking for

and if we try to look for a course that

doesn't exist for example if I add some

garbage here

as you can see we are getting a 404

status now we are also getting a trace

with it uh we may

configure our exceptional responses

later on but for now we will just leave

it like that

let's get rid of the garbage let's send

once again and yes we are getting

the course details now the thing is

the response in the course detailed

request and the course list request is

pretty much the same

I think it would be nice if we could get

the list of the lessons that belongs to

this course inside

this details response right let's do

that let's finally start working on our

lessons

so I'll come back to intelligent

I will create a new model

new

Java class and it's going to be called

lesson

this is going to be a node just like

before

it will have

a long ID

is going to be the ID and also a

generated value

uh okay so I will have to import the uh

I think there's something wrong wait 80.

yeah ID sorry it it didn't get imported

first yeah

next we will say private stream title

and private string identifier so these

are the three properties that we need

next up we will generate our empty

Constructor

and our Getters and setters

yes now before we try to implement our

relationship between the lesson and

courses let's go back to our database

and have a look at how it was mapped

if we focus on a single Corsair as you

can see this course has four lessons

and each of these lessons has a belongs

to relationship with the course

now from the point of view of a single

lesson it has an outgoing relationship

of type belongs to with its parent

course

and from the point of view of the course

it has an incoming relationship of type

belongs to with the lesson

now with that information we can

actually Implement these relationships

in our models

now back in our code we will open the

source code for the course model

and we will describe a new property

private

list

list of lesson

called lessons

plus new error list

now this list here is going to be a

relationship this is another annotation

that comes with

the dependency that we have used

and

the type of the relationship is belongs

to like we just saw in our database

and the direction

from the point of view of the course

it's an incoming Direction

or an incoming relationship so the type

of the relationship is belongs to in the

direction is relationship dot Direction

dot income

now we will need a getter

we do not need a Setter since we are not

going to set this relationship manually

we just need a getter

now since we have implemented the

relationship we have implemented the

getter it should work let's rerun the

application

come back to insomnia

make the request once again

and as you can see now the course comes

back with a list of the lessons as you

can see these Blazer shopping cart of

course as four lessons

now this is all nice and cool but there

is a simple problem

so if I go back to the get all courses

and make a request

as you can see all the courses are

coming back with the lessons

now depending on what you want from your

application this may be something that

you expect or you do not expect for

example

in some places in my application I may

not want the list of the lessons at all

because whenever a course is coming back

with the list of lessons it's actually

querying for all these lessons

so if I do not use the lessons these did

that entire query time that that that

time my server is working is going to

waste

unfortunately there is no way that I

know of to leave out a relationship from

a query

so if you describe a relationship like

that

you cannot actually lazy load it every

time you will query for the courses you

will also get the relationship back

now this is something that's not that

hard to solve actually we can write

custom Cipher queries to query for the

lessons when they are needed

and leave them out when we don't

to do that first let's get rid of the

relationships from the model

also remove this because we are not

going to use them

now at this moment in our code base the

courses and lessons are totally

unrelated to each other

but we have to implement some sort of

relationships between them

now my idea is that we can have them

unrelated in our code base but what we

can do is we can actually create another

Repository

it will be another interface

let's call it lesson Repository

it extends

the neo4j Repository

the object will be lesson

and the ID will be long

and inside this repository we will

Implement another function that is list

lesson this is going to be the return

type

find

lessons

by

course identifier

and we'll get the identifier as a string

okay now we have a method that is going

to find a bunch of lessons using a

course identifier now the course

identifier part is not a property of

lesson so we cannot have and

automatically build method like this

instead what we will do is we will write

a very simple Cipher query

so we'll say query

and inside let's begin

we'll send match

and inside a set of parentheses I will

say colon course

so we are looking for all the nodes of

type course

inside I will say identifier

identifier

so we are looking for a course node that

has the same identifier as we have

passed to this method

now

once we have found this course we will

need to get

and incoming relationship

are of type belongs to

and the related node should be

lessons lesson

and finally we have to return

the lessons

let me explain what we are doing here

first we are trying to find a course

node that has the same identifier as the

one we have passed to this method

then we are looking for all the lessons

that are related to this course

using an incoming belongs to

relationship

now once we have found all the lessons

we are putting them inside another

variable called lessons and finally we

are returning the lessons

in fact we can we can get rid of this R

variable here since that's not being

used anywhere so the only

piece of variable that we need is this

one here

once again we are finding a course by

the identifier that we have then we are

looking for all the lessons that has an

incoming relationship with this course

of type belongs to

then we are putting those lessons inside

the lessons variable and returning them

to the user so this is the query that we

have

now if everything goes fine it should

return a list of the lessons

okay the repository is all set let's

create a service

lesson service

this is going to be a service class

and inside the class we want a private

final

lesson repository lesson Repository

we will have our Constructor

and we will have

public

person list of lesson

the name of the method should be get all

lessons

by course

identifier

we will get a string identifier

and then inside the method we will say

lesson repository dot find lessons by

course identifier and we will say

identify

that's it and we can just return this

directly or return this directly

and yeah that's it that that's all you

need in the lesson service

then

inside the controllers as you can see

right now we are returning a course

now this is all right but

what would be better is if we return a

dto or a data transfer object

so I'll just rename my

config package to objects because we are

not going to use the config right now

refactor

and we will create a new class

called course

dto

so this is a plain Java class

inside I'll subscribe it string

identifier because I want the identifier

for the course

I want private string

and title

I want the title for the course

private string

teacher

and

private list

list of lesson called lessons equals New

Era list

excellent

oh it looks like we have to import the

lesson class

yes

finally we will have our Constructor

an empty Constructor for that because we

are going to insert all these values

later on

or you know what we can get all this

like this

Constructor identifier yes let's let's

take these three and leave out the

lessons because we are going to fill it

up later

next we want to create some Getters uh

we need Getters for everything and we'll

need a Setter for the lesson

yes so that's that's all we need for the

moment

now let's go back to the course

controller inside this method instead of

returning a course I will select course

tto

and I have the course here

then I I will create course tto

response course another object equals to

new course tto

first I will need course dot get

identifier

next I need the title so of course

dot get title comma finally

ports

dot get

teacher yeah

those are the three

and then I will say response course Dot

set lessons

and now we need to bring in our private

final

lesson service lesson service here

we need to add lesson service lesson

service

and inside

our Constructor and set this

dot lesson service equals less service

yeah so we have

that

initialized

this looks fine to me

now inside response course dot

setlessons I'll sell lesson service Dot

get all Distance by course identifier

and course dot get identifier so I will

get all the lessons inside this course

finally in the return statement instead

of course I will just say response

course

now technically if we have written

everything correctly this should give us

the list of lessons along with the other

course details

so I'll just restart my application

it has been restarted let's go back to

insomnia

send a request for a single course

and as it looks like we are still

getting the list of the lessons but if I

go back to get all courses

as you can see now there are no lessons

at all so inside single course I can get

the lessons because I want them to be

there

and inside get on courses I do not need

them so I don't have them at all

I hope you have understood how you can

write a simple custom Cipher query

inside Java and this is going to be

really helpful now creating this course

dto class and later on we are going to

also create query

results class

this all things may seem like a lot of

work to you but to be honest once you

have everything set up you have the code

ready to go you will see how easy it is

to make changes because

uh you are making sure that you are

going you are abiding by the separation

of concerns and things like that

so okay we have done that now one thing

that I'd like to say that the way the

front end of our application has been

created we actually need the list of

lessons in the list of courses as well

so

what I am going to do is I am going to

make some changes to it

this uh list or the

course index method and make sure that

we are getting the list of lessons there

as well

so I will change it to a course tto

and yes the return hold return statement

won't work anymore

so I will copy this

I'll send list

course

courses

ah sorry

courses

and I'm getting a list of all the

courses in our database

then

what I'm going to do is I am going to

create another list of the course tto

type

and this will be the response courses

equals

I will say

courses dot stream so I am going to use

the stream API of java

then I will call the map

now inside map

I will say course

foreign

and here I will create a new core Studio

or Studio response course

equals to

I can actually copy this entire line

yes that I can also copy this line

read it after this yes response course

dot lesson.service so now I have a new

course tto

and I need to return this so I'll say

return

sponse course

and finally

Dot

collect

character stop to list

so what I am doing here I am getting the

list of courses from my database without

the lessons

then I am mapping through that list

accessing each course inside that list

of courses

I am treating a new dto object I am

getting all the lessons for that course

and setting that inside my dto object

and finally I am returning that response

course from inside

the standup function

and once I have returned that I am

creating a new list and adding that new

response course to that list so this is

how it goes finally inside response

entity I will just replace this part

with

response courses

http status okay

let's save our code and let's restart

and see if it works fine or not

let's go back to insomnia

make a request for all the courses

and as you can see all the lessons are

back in the course class

so this far we have finally implemented

the courses we have implemented the

lessons we have written some custom

Cipher query

so I think you have done a lot of work

already to be honest

now next up we are going to implement uh

the user model we will introduce an

authentication system to this

application so that users can log in

and once that done we can finally work

on the course enrollments

so let's continue working on it

now that we have finished implementing

the courses and lessons it's time to

finally work on the user now before you

start writing our code for the user

related functionalities we will need to

add

a new dependency

so open up your pom.xml file

then right after spring dot EnV add a

new dependency

now the name of this dependency will be

spring

boot

starter

security

and its group ID will be simply org dot

spring

framework

dot boot like all the other starter

packages in fact let's do one thing

let's cut this chunk of code from here

and put it right after the starter data

neo4j that way we will have all our

starter packs in a in a line and then we

will have the other dependencies

so once you have added this dependency

to your project just

hit this load Maven changes button

okay it looks like the dependency has

been resolved

now

let's try to restart our application

and you should see something like this

using generated security password the

generated password is for development

use only and things like that

so this means that Spring Security is

actually working now to test it out we

can go back to insomnia

we can send one of our old requests

and you will see that it says 401

unauthorized which means that the spring

security system is stopping us from

accessing this endpoint now this is

totally normal by default Spring

Security

stops all the requests to the server but

you can use an username and password to

get access to the data you want now for

that let's go back to our IDE look for

your

application.properties file inside SRC

Main and resources

now after the three lines that we added

before we will have to add two more

lines these are spring Dot security

dot user dot name

and I'll set it to Farhan

spring.security.user Dot

password equals to password

now let's go back to insomnia

here you can see there is the auth menu

right besides body so I will go inside

auth I will click on this arrow and pick

basic auth

inside here I will add the username

Farhan and password is

password then I will send the request

okay it looks like we forgot to restart

our application

so let's do that

now insomnia

sent request

and it worked out just fine

but I know what you are thinking this is

not the right way to authenticate any

request we want to use the users in our

database right

don't worry we are going to do that but

before I just wanted to show you how you

can

make your request go through the Spring

Security

gate

now let's disable the Authentication

okay

let's go back to our IDE and start

working on a proper authentication

system first let's remove these two

values

close the properties stop our

application and minimize the timing

now we will start by creating a new

model

named user

now unlike the models that we have

worked on before the user is going to

implement an interface so we will write

implements

user

details

now this interface is actually part of

the Security package that we just added

and it describes a few

methods and properties that makes this

model usable for our authentication

system

so we can hover over it

and it says Implement methods we'll

click that

and it it shows us a bunch of methods we

will say okay

and now we have a lot of overrides let's

just close this sidebar for more space

okay so these are the methods that we

want to implement

so let's start with uh

get password and get username because

those are the easiest one

we will say private

a string username

private

string password

then inside get password we will return

password

and inside username we will return

username

excellent now those two have been taken

care of next we will add

the roles so you would say private

string

roles I hope that you remember from our

database view from before that every

single users has a bunch of roles so

this is what we are going to do here

now once you have added the roles

we want to implement the get authorities

method which is this one

now this is sort of and special kind of

method that actually returns all the

authorities or roles for this user

now you may have seen that I had added a

role user value to our

users in our database but you can have

something like role admin

role teacher and so on you can also have

role admin comma role teacher

role

moderator and so on but for now we are

dealing with a single role

now this method here it returns a

collection and that extends the granted

Authority interface

so what we need to do is we will say

return we need the arrays class from

java.util and then we will say dot

stream

inside the stream method we need our

roles so the roles that we have defined

here will say roles

Dot

split

and we want to split it by a comma

because remember we can have roles like

this

roll

admin

role

teacher

so we want to split it using this comma

after splitting let's come to the next

line and we will say map

we will say simple

granted Authority which is a class and

we will add new

at the end

and Dot to list

see and we are returning this list so

what we are doing here we are getting

the value of the roles string from this

user

we are splitting the string using the

commas so just in case you decide to add

multiple roles to this user

and then we are mapping through that new

you know splitted string and creating

new simple granted authorities using the

string value that we get from our map

call

finally once we have multiple or a

single simple granted Authority instance

we are adding them to a list and

returning that list from this method so

since lists are part of collections

these checks out

now if you are confused about this

method I would suggest that you do some

more study on how Spring Security works

and how you can Implement multiple roles

for users so that's a bit out of the

scope of this course so this is what we

have to do now

next up let's go inside is account

non-expired we want to return true here

then is account non-locked yes we want

to return true is credentials

non-expired true

and also is the account enabled yes true

so that's it we have successfully

implemented the user details uh

interface but other than these three we

also need

an ID

and we need a private string

name for the user

unlike the other models this ID is going

to be an ID and it is going to be a

generated value

and we will need a getter and Setter for

our name

so

let's come at the end of all these

overrides

or maybe

maybe here let's come to this place

and we'll say alt insert we will

generate a getter

for our ID

no I mean our name

we will also need a Setter for our name

there you go

now in the case of this user model we

will actually use our ID field so we

will also add a getter for that

and since we are going to create new

users we will also need Setters for our

username and password and roles

so let's scroll down

and come to get password and I will come

after it and insert

a Setter

for our password

there we have it

next after the get username I will

insert a Setter for our username

and finally we do not have a get roles

method but we have this one so I will

just come after this

and I will input a Setter

for our rows

there you go

it looks like the model is ready to be

used

and you know what let's just generate a

getter for our roles as well because

even though we have this get authorities

method we may also need a plain old

getter for our rolls string so let's

just add the getter

and it looks like everything has been

set up finally uh I I would just like to

check the channel you know once again to

see if we have anything left

yeah we have left ID Setter but we are

not going to need that

so once you are done with that let's

just close this

and we will now create a new package

called

config

next up let's create a new Repository

so I will right click on the

repositories package create a new Java

class it has to be an interface and name

it user Repository

uh like the other repositories I want to

extend uh neo4j repository and that's

all we need for now we will come back to

this because we will add two more custom

queries uh to this repository but those

are not needed at the moment

finally we will go inside service and we

are going to create an special service

for our authentication system

so you will create a new class called

Neo

user details service you may call it

something else like neo4j user details

service the thing is this is going to be

an user Detail Service that we will in

turn use with Spring Security

so we have this class

and it's going to implement another

interface called user details service

and just like the model we will

implement the methods there is only one

method load user by username

and that's pretty much it

so as you can see from the name of this

method these can return a user using its

username so for that we will go back to

our user Repository

we will create

a method that returns an optional user

and we will call it find

user sorry I have made a mistake we will

have to delete the name of the model as

well as the type of ID we are using

all right now we can say find user by

username

we will receive the username as and

string

this is very simple method and that's

all we need

now inside our new user details service

I will explain why we need this class in

a bit but for now just write the class

along with me

so I will create a private

final

user repository user Repository

then I will have a Constructor

there you go

and then as you can see this method here

Returns the type user details this is

one of the interfaces that comes with

the Spring Security framework

so instead of returning null

what we are going to do is return

sorry user Repository

dot or let's come to the next line find

user by username we'll pass the username

that we have here

and then on the third line we will say

or else throw

and inside the method we will create a

new Lambda

new

username not found exception

and we will pass a message

username not

found

then you can just

add the username to the message yeah so

that's what we have so as you can see

the user details service interface

implements a method called load user by

username so what we are doing here we

are using our repository instance and we

are calling the find user by username

method now it returns an optional user

so if we have the user then we will

return it and if we do not have the user

then we will throw a new username not

found exception and we have to throw

this exception because uh by the

definition of this interface this method

has to throw a username not found

exception in cases where the username is

not found

so that's all we need here

now finally we are going to write the

big chunk of authentication code that we

will need to use our database instead of

some preset user

so go inside project I will collapse

everything then go inside

source main Java

and as you can see I have a package

called config so I will right click and

create a new class

called security config

by the way the service class that we

created has to be annotated with the

service annotation yeah

that's all the service classes have been

annotated okay so inside the security

config class we have to annotate it with

configuration

then enable

web

security

okay then inside the class

we will create a private final

Neo user details service

we will have a Constructor

and then we are going to create a bean

and that b is of type security

filter

chain

new security filter chain

it will receive HTTP security

this one

we will call it HTTP security and these

throws

an exception

okay let's annotate with the bean

annotation so that

neo4j will handle the task of

instantiating this class and everything

now inside this method we will create a

new security filter chain once we have

finished working on it you will

understand what it actually does so

we'll say return

HTTP security this one

then dot session management

inside will say session

get a new Lambda

session Dot

session

creation policy

then session creation policy dot

stateless because

since we are creating an API the

authentication has to be stateless

then we are going to disable csrf

protection because

we don't need that

for that we will use abstract

http

configurer yes this one

and noticeable

okay then on the next line we will say

course because we are going to customize

our course

customizer

dot with defaults so course will be

configured with the default settings

finally we'll say authorize HTTP

requests

inside we'll say auth

and auth

Dot

it will say request matches yes

and inside the request matches we will

say

API

slash V1

slash auth

slash me

okay

let's leave it to that for now we will

add more routes to it later so I'll say

dot authenticated

and Dot

any request

Dot permit off

finally

we will finish this by saying user

details service

Neo user details service

dot HTTP basic

customizer dot with defaults

and

dot build

Ah that's a lot of code right so let us

take a look and try to understand what's

happening here so

by default the Spring Security framework

uses a security filter chain that blocks

any request to your server

you will have to authenticate for every

single request to the server and we do

not want that

also in the default security filter

chain it uses your

application.properties file or a

randomly generated user as an

authenticatorable user we also do not

want that we want to use the users in

our database

so for that we will need to create a new

security filter chain that will

essentially Define how we want to secure

our application

so inside this pane I hope you

understand what beans are beans are just

Java classes that are managed to buy the

spring boot framework so we are creating

a new Bean

that is of type security filter chain

and it throws an exception

okay so inside this bin we are saying

return HTTP security which is of type

HTTP security

this is

one of the classes that comes with the

Spring Security framework

and the first thing that we are doing is

we are making the session stateless I

know this is a long line but all this

does is it makes the sessions stateless

so we do not hold any session data

whatsoever

then we are disabling the csrf

Securities because since it is an API

and we essentially do not have any web

interfaces within this particular

application or service we can

successfully disable the csrf protection

then we are configuring course so in an

application where there is no spring

boot security framework you have to

customize your cost settings somewhere

else but since we are using spring boot

security this framework is going to

intercept the request first and then it

will pass it to the application or other

configurers so we have to configure the

course right in the security filter

chain configuration thing

so right now we are

customizing the cost settings with the

default ones but we are going to change

it very soon

then we are saying authorize HTTP

request so this is the method where we

will Define which routes has to be

protected by our application and which

routes do not need to be protected so

inside we have an auth object and we are

using the request matches method to say

that hey Spring boot security you have

to protect any request

to this particular endpoint we haven't

created this endpoint here and you only

need to stop any request coming to this

endpoint and all the other requesters

any requests

just permit them to connect

so

when we get a request in this particular

endpoint it needs to be authenticated

any other request

should be permitted

now next up we will actually configure

our course configurations so we will say

pause Pawn figure

or con course configuration sources this

one we want this one

and then we will say course

configuration Source we will start the

method and we'll annotate with a beam

now inside this bin we will say

course configuration

course configurations equals to new

Parts configuration so we are creating a

new course configuration

then we are going to

change things in this configuration so

we will say course configuration Dot

set allowed

Origins

so which Origins are we allowing so I'll

just say

array let's check what it takes so it

looks like

it takes a list of strings so I will say

array

dot as list arrays.list

and I will create a string http

localhost

3000.

and also just in case http

127.0.0.1 which is localhost as well

three thousand so I have been made aware

by Gavin long that the react application

that he will be working on

will run on localhost 3000 or 127001

Port 3000 so what I have done

I have allowed these two Origins to

connect to my to my application

now this is one way of doing things to

be honest these two values these should

be resided inside our DOT EnV file but I

am not going to do that since I have

shown you how to use a DOT Envy file I

will just add a to-do here

which says

make sure that

the origin list comes

from and in

environment five so I would suggest that

the viewers or you essentially take this

as a challenge and implement this part

using an EnV file put your code on

GitHub and share the link in the comment

section I will take my time to go

through every single code shared in the

comment section and I will make sure

that everything is right so this is one

of the challenges I will add that as a

to-do and you can actually click on this

button here to do and you can see

what challenges I have set

so that's one challenge

then the second one is course

configuration dot set

allowed methods and what methods are we

allowing

areas dot as list this also takes a list

so

this is how you are going to do kth and

we will do Post

we will do patch

then put

delete

options

and

these are all owed methods then we'll

say course configuration dot set allow

credentials yes this one and with set

two

nice

then we will say course configuration

dot set allowed

headers

and arrays

dot as list

authorization

request

type

content

type so these are the three headers that

we want to allow

finally we'll say configuration

dot set exposed headers

and it will be called arrays.list

inside here we will say

X

get

header

and lastly we will have course

configuration dot set max age

we will let's say let's do it 360 3600l

so L for long next less let's just

create an URL based course configuration

source so we'll say

URL paste course configuration source

and URL based on course configuration

Source equals to new

URL based course configuration search

here try saying that 10 times faster

right URL based course configuration

Source yeah I can do that

and we will say URL based course

configuration Source yes

dot register course configuration

and we want to apply these configuration

to all the routes in our application so

that's what this pattern means so this

is the root and anything that comes

after that

comma and course configuration yeah

nice finally we have to return

our URL based course configuration

source okay so now we have our course

configured one last thing that we have

to configure is actually a password

encoder so I believe you remember when I

showed you our database architecture

that we are saving the password for the

users

as Big Crypt encrypted hashes we are not

saving plain text passwords because you

should never do that

so we will need to let the spring boot

security know what encoder we are using

so to do that I will say password

encoder

password encoder

this is another Bean

I will annotate this one with pin

and then inside the method I will say

return

new be creeped password encoder

and end it there

so yes we have done most of the things

oh by the way I think I have missed

these three lines so what we are saying

here

is we need to let the security filter

chain know which user details service

are we going to use and in our case we

have a custom user details service which

is the new user details service and what

we have inside this class

we have implemented a single method that

helps us to use or look for the user so

every time some user tries to log into

our application the Spring Security

filter chain will use this detailed

service look for the user and then

check the password against the hash that

we have saved in our database using this

password encoder and if it matches it

will let the request pass through and if

it doesn't it will throw a username not

found exception

so finally we are going to use an HTTP

basic authentication I know this is

pretty insecure but for the scope of

this project or this application this is

totally fine I could have implemented a

a more sophisticated JWT based

authentication but that will take a lot

more time and a lot more code so I want

to focus more on the on the neo4j and

the spring boot connectivity rather than

the authentication and security part of

the application so this is what we are

going for

so now that we have configured our

security thing let's let's just go ahead

and create this

um

end point so we will go to Project

and we will create a new controller

and it will be called user controller

let's annotate this one as

our rest controller and request mapping

to

apis slash

V1 slash Alt

inside we'll have a single method for

now

which will be

public

logged in user

and

we will create an end point which will

be a get mapping

and it will be called me

finally

inside this logged in user we have to

Define our return type which will be

string for now

and we will get something called the

principle I I will explain what

principle is and we will return

principal dot get name

so okay let's make sure that this

endpoint is right so let's go back to

our config

API V1 auth me

so everything checks out

hmm

okay then let's start our application

and try to log in using one of the users

in our database

right now we are inside our neo4j

dashboard and we can see all the users

in our database so I am going to just

select John price and the username is

just price

now the password for all these users

such as Mirza Mikhail or MacMillan the

password is actually password I have

already encoded the password using Big

Crypt and saved the hash

so I will just copy the username of John

price price and let's go back to our

insomnia

then right now if I make a request

without any Authentication

it should work because the single course

and get all courses endpoints have not

been protected I hope that you

remembered that the only endpoint that

is protected right now is the me and

point so I'll create a new HTTP request

I will call this me

HTTP

localhost 8080

API slash P1 slash

auth slash me

and if I send a request as you can see I

am getting 401 unauthorized but if I go

inside auth select basic auth and as the

username I will say price

price and for password I will save

password let's see if it works or not

so as you can see this time we are

getting 200 so what's happening here

let's go back to our code and let me

explain it to you

now right now we have a single endpoint

called slash API V1 auth me and

according to our security configuration

this is one of the protected route and

we have to be authenticated to hit this

route

so inside this method we are getting a

principal object and this principal

object

usually contains information about what

user is authenticated right now so once

you send in your username and password

our security config will intercept that

and pass that to the user Detail Service

the user details service in turn will

look for an user using the username that

we have got from the user and if it

finds the username it will give it back

otherwise it will throw a username not

found exception

once the filter chain has found an user

it will get the password from that user

and

check its hash against the password

submitted by the user using this

password encoder

and you may ask how does the filter

chain gets the password well if I go

back to models user and you will see

we have a getter for the password get

password and this is a part of

the user details interface so that's how

it gets the password hash

and if the user is giving the right

username and password he will pass

otherwise he will not pass so that's how

this entire authentication system works

and if you have done anything right you

should be able to authenticate new users

to your application

now that we have our user we can finally

begin working on the enrollment part so

if user can actually look at our courses

and enroll in the ones that they like so

that's what we are going to do now

now that our users can log into our

system we want to create a way for them

to register in our system so we have a

way to log in now we need a way to

register

so before we start writing the code for

registering the user I would like to

create a few packages and classes

now the first one

is a package called requests

and inside here I will create a new Java

class called

create user request

now I will explain what it will do but

just follow along for now

then inside this class we will say

private string name

private stream username

private string password

private string rows so everything that

the user model has except the ID field

next up we will create a Constructor and

this time we will pick all of them

and after that I will just create some

Getters and setters

okay the request class is ready let's

close that

next what we need is a user dto object

so user dto

and this one is going to be really

simple

private

stream name

private string username

private string

rolls we do not want to return the

password in our responses so we will

just leave that out

next we want to create the Constructor

and the Getters and setters

there you go we have a user dto now

next we need to create a service

called

user service

user

service

and this one has to be annotated as a

new service

for sure

then we will need a private user

repository user repository let's let's

make it final

and then we will have

private final

password encoder password encoder

because we want to encode our user's

password

then I will have my Constructor

and a function

public

and this is going to return a user

create user

and this will take

a create user request not any request

but create user request

and create a new user

so I will say user

user equals to new user

then I will say user dot set name

request dot get name

and I hope you remembered that we just

implemented this get name method inside

the create user class so that's why we

can use that

then user dot set

username

request dot get username nice

user dot set roles

request dot get roles

user dot set password and this is where

the encoder will come in and I will say

password encoder dot in code

inside there I will say request dot get

password so this will take what the

password is that we got from the user

and then encode it and set it as the

user's password

finally we will say user repository dot

save

user the safe method comes with all the

repositories and we will return the user

that's it we have our service method

ready to go

let's go to the controller now

and here

I will say

probably

and then response entity

of type user dto

sign up

and this will take

a request body and map that to create

user request

we'll call it request

and let's go inside the method and I

will say

oh I will also need

a reference to our private final user

service

user service

and I need my Constructor

then I can say

user service dot create user and I can

just pass the request to it

and it returns a user

then I will create a user dto response

user equals to

new user dto now first I need the name

so user dot get name

then user dot get username and user dot

get rules that's all I need

then I can just return a new response

entity

and the body will be the response user

and

the status will be created because we

have just created a new user

so yeah that's all we need to sign up a

new user and this is really easy to do

uh I will have to create a post mapping

or

register

yeah that's all we need now we can go

back to insomnia and check if this works

so I'll just start my application

then I will switch to insomnia

and I will create a new request Alt

register

okay this will have a body

first we need a name

so I will do Farhan

Singh children

comma I need an username

Farhan

then I need a password

and I will need a roots

string so it will be rule

User make sure your

writing the role like this

and then I'll send the request to http

localhost API slash V1 slash auth slash

register

that's it I believe

okay let's change it to post

and we do not need any authentication

for this and let's hit send

so looks like a new user has been

registered it says 201 and the username

is Farhan now there are two ways to

verify if the registration has gone

through correctly or not first I can go

to my me request

I can go to basic and instead of price I

can save Farhan and the password is

password

I'll say send

and it looks like it gives me back

Farhan which means the user has been

registered to database but still if we

quickly switch to our database dashboard

then

click on user once again

it got me back all the users and as you

can see Farhan hasin show 3 is right

here but right now Farhan or me I am not

enrolled in any of the courses so next

we are going to work with the enrollment

process and we are make we are going to

make sure that all the users can

actually enroll in a class but before

that I would like to give another

challenge for you so if I go back to the

user surface as you can see right now I

am not checking if this username has

been already registered or not

so I will leave it to you

so that you can figure out how to check

if a thing exists on the database using

stream boot data or a spring data

repositories and make a decision on that

so I will add another to do and I will

say

makes sure

that these

username does

cyst

and let's just cut that from here and I

will put that over the username so this

is another tutu that I am leaving out

for you I will look forward to any

submission in in the comment section and

I will actually let you know if anything

is wrong with the code

so let's close that

and start working on our enrollment

process

now before I write anything in my

services or in my controllers I would

like to make sure that we have all the

necessary queries in our repositories

for example in order to enroll a user in

a new course we will have to create a

new relationship between the user and

the course

so like I showed you before you can

actually do that through the models but

in this case I am going to create a

custom query that will have a custom

query result and you can use that to

create a new enrollment relationship

so for that

I will go to my repositories and then

user repositories

and I will write two methods here the

first one is going to be a Boolean

and it will be called find enrollment

status

so you can use this method to check

whether a user is already enrolled in a

course or not

now I'll take a string

username and I will also take a string

identifier which is the course

identifier in our case

so we are going to write a custom query

just like we did before

and it will go something like match

user user

or this has to be a string of course

comma

there will be a course course

and then fire

user dot username

equals dollar sign username so this

variable right here

and

user sorry course dot identity fire

equals dollar sign identify right here

this variable so what we are doing is we

are querying for a user

that has the same username as we have

passed to this method

and then we are looking for a course

that has the same identifier as we have

passed in this method and we are saving

the user in a variable called user and

the course in a variable called course

next we want to give a space and then

enter so that we can begin on next line

and then we'll say return

exists

we will say

user

enrolled in

yeah that's pretty much it so what we

are doing is we are looking for a user

and course against the username and

identifier that we have and then we are

checking if there is any outgoing

relationship from the user with the

course of type enrolled in so I believe

you remember that back when we are

designing our database we had a

relationship called enrolled in and it

was the outgoing relationship from the

point of view of the user and an

incoming relationship from the point of

view of the course

so there we go we can use it use this

method to check whether a user is

already enrolled in a course next up we

want to create a query that we can use

to create a new enrollment now before we

do that we need another class

so you create a new package

query

results

and inside that we will create a new

class

called

course enroll

ment

query request

or query results sorry

pretty result now this is a very simple

class we will say private

user user private course

and we need to import these classes

obviously

uh yes

okay we have both the classes then we

will have

and

empty Constructor

like that

and we will have some getter sensors so

alt insert

getter answer both of this

so there you go we have a course

enrollment query result object

now if I come back

and I create a new method

and this is going to return our course

enrollment query result

I would call it create

enrollment

relation

ship basically you want to be explicit

with your names

and I will need a username

also I will need an identifier

now the first bit of this query is

identical to this one so what I am going

to do is I will copy this entire match

part

come inside this method and it's a query

put it in space and then enter now on

the next line I would like to say create

user in fact you can just copy this this

part here

so we are going to

we are going to create

a new outgoing relationship from the

user to the course of type enrolled in

then finally you would like to return

the user

and the course

now as you can see this query returns a

really complex data it returns an user

object and it also returns a course

object so like other custom queries like

this one we cannot just say that these

returns are Boolean type

instead we had to create our own custom

class that has an user object and a

course object and use that as the result

of this query so this is how you can map

complex queries to custom made classes

okay so our user repository is now ready

to go I believe we can enroll our users

to classes now

but we cannot just put the code for

course enrollment inside the user

service or user controller or the course

controller at all instead what we are

going to do is we will create a new

service called

course enrollment

service

and it will be a class

a service class

and inside there we will have a few

methods

now the first method that we have

but before that we will need a private

final

course Repository

course repository and private final

user repository user repository now this

service can access both repositories

because we are going to work with both

courses and enrollment

but if we had to put this code inside

the core service then we would need to

unnecessarily refer to the user

repository inside the course core

service that doesn't

feel good at all that's why we have a

separate service for you know creating

the enrollments

so inside we will have a Constructor

and we will have a few methods

now the first method would be public

Boolean

get enrollment

status

and for that we will need a username

and course identifier

then inside the method

we will just say return

course repository Dot

check uh sorry

user Repository

Dot

find enrollment status and we will pass

it the username and the course

identifier so this way we can use the

this method to check if a given username

is enrolled in a course or not

now the next thing we need is

public and this is going to return a

course enrollment query result

and it will be called enroll in

and it will also need a username

and course identifier

okay we will return

user Repository

Dot

create enrollment relationship

username and identifier okay

so yeah we are done with these two

methods we will need one last method

that is

we need public

list of courses

and we will call it

get all enrolled

course by username

string username and string identifier

okay so what this method is going to do

is if we give it a username

it will give us back the list of courses

the user is enrolled in and I just

realized we really do not need the

identifier here

now we do not have a repository method

for this so let's go to our course

Repository

and create a new method

this will return a list of courses and

we will call it find all

enrolled courses by

username

and we will receive I stream username

and then we can say query

match

we are trying to match a user

where the username

is username so this is another way of

doing a wire query or where clause

so we will have a user with the same

username that has been passed to this

method

then once we have that node we want to

get

enrolled in

courses course so we want to get all the

courses that has an outgoing

relationship with this user of type

enrolled in and I am saving all those

courses in the courses variable and I'll

just return

courses

and we will also import the list method

java.utl of course

and yeah that's it so let's go back here

and we will say

return

course repository dot find all enroll

course by username and we'll just pass

the username so that's it we have our

course enrollment service ready to go we

can use it to enroll our users in a new

course

we can get the enrollment status of a

user and we can also get all the

enrolled courses by

username

and I just realized it should be courses

and not just course

so there we go

okay now I think we are ready to create

a new controller

the controller will be named course

enrollment on

roller

like the other ones it's going to be a

rest controller for sure okay let's just

go back to our service class for a bit

course enrollment service and here we

have annotated it yeah and we will do a

request mapping

API slash V1 slash

enrollments

now these enrollments

endpoint these need to be protected

because if you go back to our service

for example course enrollment service

you can see all these methods needs to

know the username of the currently

authenticated user so for that we need

to protect this route

so we will leave the controller open and

get back to config and security config

let's let's close this

and come to the request matches part I

will put a comma

and I will say

API slash

V1

slash enrollments

slash double star which means

anything that comes after API slide V1

slash enrollment slash

anything should be authenticated or

protected let's close that

and inside our controller class

we will need a few services

so I will say private

final

course enrollment

service course enrollment service

and we will also need private

final

lesson service

lesson service because we will need to

know what lessons are part of a certain

course

now I will have my Constructor

and we can start working on the

endpoints

now the first one I want to do is uh I

want to let the user enroll in a course

so for that I will say

public

response entity of type

course in

enrollment dto

and we need to create this class create

class course enrollment tto and we want

to put it inside

the objects packets

and here we have our dto

we need string

private string username so the user

name of the user that has just enrolled

in the course private history name of

the user private

course

and the course that the user just

enrolled in

then we will have

a Constructor

and we will have the Getters and setters

yeah that's pretty much it

so you want to return a course

enrollment dto and we will call our

method

enroll in

now what we are going to take we are

going to take the request body

request body and we will create another

request called

course enrollment

request

let's hover over it and create type

parameter course enrollment request

uh you know what let's just copy the

name

go to Project

go to requests

create a new class

called course envelopment request

and inside this we need

private

string identifier this is the course

identifier of course

and we will

generate a Constructor

and the Getters and setters

okay let's go back to our code request

body course enrollment request so we

will have our course enrollment request

comma

another thing that we need is the

principle sorry I forgot to name this so

I will get the principal principle

because this is how you get the username

of the authenticated user now inside

this method we will say course

enrollment query result

let's call it enrollment quiz result yes

equals

we will say course enrollment service

dot enroll in

what is the username the principal dot

get name and what is the identifier

request dot get identifier

we have a new enrollment request query

right query result now we cannot just

return this query result for button

or verbatim rather we'll need a new dto

called

course enrollment TT

now I will over it

some more action

okay let's again just copy this go to

Project

go inside objects

oh it looks like we have a course

enrollment video grid I just forgot

about this

so what I will do is course enrollment

dto

we have that yeah I just forgot that we

created this

so response enrollment

equals new

the first thing that we need to pass is

the username so we will say enrollment

query result dot get

user dot get username because our query

result has a git user method

that in turn returns a get username

method and we can use that to get

the name of the user or the username of

the user next up we have the name of the

user so enrollment query result dot get

user once again

dot get name and you know what this line

is getting too big so I will just put it

in the next line

comma that's bring down here yeah much

better

enrollment query request Dot

get course

that's pretty much it all we need the

username name and the course

finally we can say return

uh

new response entity

the body will be the response enrollment

and the HTTP status should be okay but

it really depends upon you since we have

created a new relationship you could

have also returned created here but I'm

going to just return okay

now this is going to be a post mapping

let's try and enroll our user in a

course okay

let's check our request once again we

just need an identifier so that's what

we are going to do

I will go back to my

insomnia client

I would create a new request called

enroll

this is going to be http

localhost 8080

slash API slash C1 slash enrollments

and inside this has to be a post request

by the way and inside the body

we need

an identifier

so let's go inside get single course

this one

go back to enroll

and we put it here

let's click on send

I just remembered I didn't restart my

code

so let's go back to idea and do that

and I will send

nice we are getting a 401 unauthorized

response because we are not

authenticating any user so I will go to

basic auth

I'll add Farhan and password is password

let's send the request

so it looks like that we should leave

the course enrollment request as a empty

Constructor instead of taking the

identifier as one of its parameters

because

if we do that the request may not work

so I am keeping it empty

and now when I try to create a

enrollment from

insomnia it just works as you can see

Farhan has been enrolled in Blazer

shopping cart application course so to

verify we can quickly go to our rdb

dashboard and if we click on enrolled in

we should see that Farhan haseen

Chaudhary is now enrolled in the Blazer

shopping cart course

now other than verifying from the

database itself we also want a way to

get the list

of all the courses a user has been

enrolled in

so we have already created a service

method for that so let's just go back to

our controller and Implement that

so here inside our controller we will

create a new get mapping method

so public

response entity this is going to be a

list

of the course tto

and we will call it enrollments

we will take the principal because we

need the username

and inside this method we will say

list course

so courses equals course

enrollment service dot get all enrolled

course by username and we'll say

principle dot get name

now we have a list of courses that the

user has been enrolled in

on the next line we will say just like

before

a list

of course dto

and

response courses equals courses Dot

stream dot map

inside the map method

we will say course this is a Lambda

of course sorry

course t t o

response course equals

new course dto

and we need an identifier title and

teacher name so I will say course dot

get identifier

course dot get title

and course dot K8 teacher

and then I can just

return response course

and finally

after this curly brace and the last

parenthesis I will say collect

collectors talk to list so this is just

like we did before with the courses list

so I'm not going to again explain this

and I will say return new response

entity

response courses comma HTTP status dot

ok

all right we this is a get mapping of

course

there you go

let's restart our application

go back to insomnia make a copy of the

enroll

request and call it enrollments

turn it into a get request get rid of

the identifier make sure that we are

authenticating

and let's hit send

okay so looks like we have a list of the

courses that this user has been enrolled

in

what we need is we need the list of

lessons too so let's go back to our

controller

I am going to steal some code from

the course controller class

I hope you remember inside this one we

actually made use of the lesson service

so I'll copy this line

I'll go back to course enrollment

controller

and put it down here

course enrollment.setlesson

and then return the response course

we will rerun the application

we'll go back to insomnia resend our

request

and this time we are actually getting

the list of all the lessons as well now

that's pretty much it we have done a lot

with our applications except I want to

implement one last thing

in in our code

so I you may remember that we have

implemented the code necessary for a

service and repository method that

returns a Boolean based on whether an

user is enrolled in a course or not so

in our list of courses I would like to

add a new Boolean variable here that

says enrolled in and that will return

true if the user is enrolled in a course

and false in case he or she is not

enrolled in a course so let's go back to

our code

what we can do is

we will go back to our course dto

and inside we want to add a new Boolean

so private

Boolean

that's use a primitive type and is in

enrolled that's it

now

we will need a getter and Setter for

this one

foreign

yes we got that

now let's go back to course enrollment

controller

and we will say

response course dot set enrolled and we

are going to hard code it to True since

the user just enrolled in this course we

know for sure that this is going to be

true so we will not make another

database request and just write it as

true

but

inside our course controller

we will actually have to do a database

call

for that we will take the principle

and we will add a check here

we will say if

principal

is not null

then

response course dot set enrolled

inside we will say

we'll need another service private final

course enrollment service

enrollment service and we will say

course enrollment service

that's a lot of politics do we must

course environment service yeah so

inside set enrolled we will say

course enrollment service

dot get enrollment status and principal

dot get name

and

course dot get identified

so we are checking whether the user has

enrolled in the course or not now why we

are checking for principle is null or

not I hope you remembered that in our

course index anyone can see the list of

our courses because it it's an open

directory so in case we do not get any

principal or authenticated user we will

just set it to false is set enrolled and

if we get a principal we will set it to

whatever the database returns so I will

rerun the application and show you how

it works

let's quickly switch to insomnia and get

all courses

so as you can see it says enrolled false

in all of these because I am not logged

in but if I log into my account Farhan

password

and hit send

you will see that it says true on the

Blazer shopping cart application and

falls in all of the other ones

now why we are here I thought I will

make one little adjustment to our code

that is in enroll

we are sending something called an

identifier now this is not clear I think

what would be better if we could call it

course identity fire

this is a lot better I think so I will

go back to my code

go inside the course enrollment request

right click on the identifier and go

inside refactor

rename

so instead of identifier I would like to

call it course identifier

I'll hit enter

yeah these are all the references

get course identifier question yes we

are going to

let's make sure that these are right

yeah let's hit okay

let's rerun our application and see

everything works fine or not

okay it looks fine

so that's pretty much it we have

successfully implemented the ability to

enroll in a new course the user can get

the list of enrollments the user can

register the user can sign in and

see the lessons in the course

now on the next part of this course

Kevin lawn will walk you through the

entire process of creating and react app

based on this API he will show how you

can use the identifiers to get the

corresponding YouTube videos and how you

can basically create a course management

system front end for yourself powered by

my API

now one last challenge that I would like

to leave for the user

is inside the services and course

enrollment service

before we enroll our user in a course we

need to

make sure that the user has

not been enrolled in the course already

so this is something that we have to

make sure right now if you try to enroll

the same user twice in a course you will

create two relationships to the same

course so it will get you will get

double results

so what you should do is you should

check whether the user has already been

enrolled in the course using the can

enrollment status maybe I don't know I'm

giving a hint

and then you should enroll the user in

the course or return something else now

this is another challenge that I am

leaving up to you so if you go inside to

do you will get three to-do's

and one is in the config and two is in

the services so make sure that you are

completing your challenges it will give

you a lot more confidence to be honest

and you will learn a lot from your

researches make sure that you leave the

link to your repository in the comment

section below I will come there I will

check the course submissions and

I think it would be a nice conversation

between us so I hope that throughout

this entire course or this part of the

course you have learned something about

the neo4j database

which truly is an amazing technology you

have learned how to create an API

powered by neo4j on Spring boot and a

few other things here and there on how

to structure your application how to use

an IDE and things like that

so if you ever need any help just give

me a shout out in the comment section I

would love to help

and best of luck for the rest of the

course Gavin long is an amazing

instructor and I can say for sure that

you will have a great time

so until the next time

take care and be careful bye

before we get started with coding this

react application I need to let you know

that I've had to make a few assumptions

regarding the knowledge and experience

of the viewers of this part of the

course

so ideally the viewer of this part of

the course should have at least a basic

knowledge of react JavaScript HTML and

CSS

so let's start with the very Basics we

of course need a code editor for writing

our react code

I will be using visual studio code for

Windows for this purpose

you can download visual studio code for

Windows from this URL

in order to run our code locally we need

to install node.js

you can install node.js from this URL

node.js is an open source cross-platform

JavaScript runtime environment

we need a local working directory to

house our applications code

so please create a working folder in a

location of your choosing on your

computer

and let's give our working folder a

sensible name my working folder for this

project is curriculum mngt management

client so curriculum mngt client then

let's open the working directory for our

project within Visual Studio code note

we can check what version of node we

have installed while we're in Visual

Studio code we can do this by first

launching the terminal window

launch the terminal window by pressing

Ctrl and the tilde character

then add the command prompt type node

Dash V and press enter

great and we can see what version of

node we have installed on our computers

an easy way to set up a react project

is to use the create Dash react Dash app

command

we can use this command to generate the

infrastructure for our react application

when I say infrastructure I mean the

node modules that are necessary for

harnessing foundational react

functionality is generated through the

use of the create react app command

there are other ways perhaps better ways

of creating the foundations for a react

project like for example using a

technology named Veet for this purpose

if you are interested please read more

about Veet at this location

using the create react app command

provides a simple way to create a react

project so I'm going to use the create

react app command for this purpose

add the command prompt type npx create

react app followed by a sensible name

for your react application I'm going to

name my react application curriculum

dash mngt mngt of course stands for

management then let's press the enter

key this process may take a while so

this will give you an opportunity to

contemplate life the universe okay you

don't have to do that but you may want

to get yourself a cup of tea or

something like that while you wait

okay great

and you can see that the foundation for

our react project has been generated

the node modules directory has been

created which contains all the react

dependencies and packages needed to

build and run a react project

as you can see a new root directory for

our project has been generated that is

of the name that we gave our application

when we ran the create react app command

so for better Clarity let's open another

instance of Visual Studio code where the

newly created project directory is the

root directory within Visual Studio code

to do this let's make sure the terminal

window is launched remember you can do

this by pressing the control key and the

tilde key

then at the command prompt

make sure the current Project's working

directory is the current directory at

the command prompt then type CD space

and then the root directory

of the react project then press the

enter key

now at the command prompt

type code followed by a space and then

the period character then press the

enter key

this will result in a new instance of

Visual Studio code being presented and

now the react project folder is the root

directory

now let's close the previously launched

instance of Visual Studio code

the create react app command has

generated the infrastructure for our

react project but has also generated a

few files that we don't need so let's

delete the following files that we don't

need report web vitals.js

setuptests.js and app.test.js

foreign

we must also delete the eslint config

setting from the package.json file here

let's save our changes

then lastly let's open the index.js file

and delete the code regarding report web

vitals

great

so we are now ready to develop our react

application so at the moment we have

very basic functionality that has been

generated for us by default through the

use of the create react app command

so let's ensure that we start off on a

good footing as it were

this lets us know

that all is working as it should be

before we start developing our

application

so to launch the react app in our

default browsers we can type npm space

start at the command prompt and press

the enter key

great and we can see the react logo

spinning which means we are good to go

so let's start creating the code for our

application within the SRC directory

where the source code will reside let's

create a subdirectory named components

as you have probably guessed we are

going to create our react components

within the components directory

let's also create a directory within the

SRC directory named images

please download this image from this

location on GitHub and ensure that this

image file resides within the images

directory we are going to use this image

as our hero image on our home page so

this image will be referenced from

within the home component the first

react component we are going to create

is the home component this component

denotes the page that will first be

displayed when our application is first

launched

so let's create a directory within the

components folder named home the first

letter of this folder name should be in

lower case

within the home folder let's create a

file named home.js

the first letter of this file should be

an uppercase

so we can use a shortcut to generate

boilerplate code for our home component

to do this type rafce like this within

the code editor and press the enter key

note that in order to utilize the rafce

shortcut you must first install the

appropriate react Snippets extension if

this extension is currently not

installed this installation can easily

be done through the extensions tab

within Visual Studio code

at the top of the home.js file let's

import the hero image from within the

images directory like this

let's include an image tag within the

jsx code return from the home component

then let's open the app.js file let's

import the home component into the app

component with this line of code

let's delete the code that we don't need

but ensure that we leave the div root

element that contains the class name

attribute that references the app.css

class

then let's appropriately reference the

home element by including the home tag

within the div parent tag like this

let's type npm start to fire up our

application

great but the image looks huge and is

not displayed appropriately for our hero

image so we are going to use bootstrap

to style this hero image so to install

bootstrap type this command at your

command prompt and press enter

to have access throughout our

application to relevant bootstrap

classes we must include this import

statement within the index.js file

now within the roots div tag of the home

component let's include the class name

attribute and set it to the bootstrap

class named container like this

let's run our code

What's Happening Here is the image is

still too large for its container

element

so to remedy this let's create a scoped

CSS style sheet for the home component

so let's create a file within the home

directory named home.css within the

home.css file let's create a reference

to the relevant image within the

container div tag like this and let's

set the image width to 100 percent

within the home component let's create a

reference to the home.css file with this

line of code

let's type npm start at the terminal

prompt to launch our application

and that is much better great

so that is a great start but we are also

going to need some other third-party

packages dependencies in order to

integrate certain functionality into our

application for example we want to

install the react router Dom package

that will be instrumental in the Global

Management of our menu and Link

functionality our navigation

functionality

we are going to use the react player

component for displaying videos that

denote lessons for our courses to our

users

we'll use axios to manage our client

interactions with a server-side web API

component so here we have used

bootstraps container class to house our

hero image so that it is appropriately

displayed within our home component as a

hero image

we can also leverage bootstrap by using

components that reside within the react

bootstrap package we'll look at using

such components a bit later in this

video

for now let's install react bootstrap to

install react bootstrap type npm space i

space react Dash bootstrap and press the

enter key

great

let's install the axios library

we are going to use axios for making our

HTTP client requests to the relevant

server-side endpoints

so as to return the relevance data from

the neo4j database that hosts the

relevant curriculum data that we want to

display to our users we'll also use

axios

to implement the enroll functionality

I.E this is functionality whereby a user

can enroll in a particular course

we could use the windows.fetch function

which is available by default within our

browsers for this purpose instead of

axios but axios provides us with key

advantages please navigate to this URL

if you'd like to learn more about these

key advantages

this link is available Below in the

description of this video

to install axios type npm space install

space axios and then press the enter key

the next packages I would like to

install allow us to leverage font

awesome from within our react

applications so at the command prompt

please type npm space i space at Fort

awesome

forward slash react Dash font awesome

and then press the enter key

let's install one more font awesome

related package

to do this type npm space i space at

Fort awesome forward slash free dash

solid-svg dash icons and then press the

enter key

great

we'll look at the details of using the

relevant font Dash awesome functionality

a little bit later in this video

the next component I want to install is

react player

this will allow us to play YouTube

videos from within our react application

the individual lessons within the

courses in which our user is enrolled

will be presented to the user as videos

these videos reside on the YouTube

platform

so to install react player type npm

space i space react dash player at the

command prompt and then press the enter

key

then let's install react router Dom

which allows us to leverage Rich

navigation and routing functionality

from within our react application to

install react router Dom

add your command prompt please type npm

space i space react Dash router Dom and

then press the enter key

if we open the package.json file you can

see the relevance dependencies that we

have just installed great

so let's create the navigation bar that

will appear at the top of our

applications UI so this can be referred

to as the header for our application

the navigation bar will contain a

register and login button

and relevant navigation menu options

so let's create a folder within the

components directory named header

please ensure that the H in header used

to name our folder is in lower case

let's create a file for the header

component and let's name it header.js

where the H is an uppercase

so we are going to use Font awesome in

this component to display a graduation

cap as the logo for our application

which will appear in the top left hand

corner of the user's screen

so let's write code to import the

relevant font awesome components

then we are going to use react bootstrap

for our buttons so let's import the

button component from the react

bootstrap package

with this line of code

we are going to use the container

component for layout purposes

let's import the nav component from

react bootstrap

which will be used for layout and

styling purposes let's also import the

navbar component for the purpose of

styling our navigation display lastly

let's import the use navigate Hook and

the nav link component from the react

rooted on package so that we can

Leverage The relevant react router Dom

functionality for navigation purposes

let's change the div tag to be a navbar

tag within the navbar element let's

include a container element let's

include the fluid keyword within the

container element

this will ensure that the navigation bar

spans the entire width of the web page

rather than placed centrally within the

web page let's reference the

navbar.brand react bootstrap element and

set its class name property to the menu

Dash link bootstrap class like this

then let's include the graduation cap

font awesome icon like this as a child

element of the navbar.brand element this

icon serves as our applications logo

let's include the navbar.toggle element

here

which will allow the navigation to be

shown and hidden through a toggle

mechanism made available to the user on

smaller screen sizes so as you can see

bootstrap is great for responsive

layouts

so let's include the nav element and

let's style the nav element

appropriately

let's include the button component

to represent our login button

the variant property is set to info as a

style choice

let's set the class name property to a

bootstrap class named me-2 this is so

that our buttons are spaced

appropriately

let's include the register button which

will appear to the right of the login

button on our navigation bar and let's

dial the register button in the same way

that we have styled the login button

right let's open the apps.js file and

firstly make sure that we are importing

the header component into the app

component

then let's include the header element

reference just above the home tag within

the jsx code that will be returned from

the app component

let's use npm start to launch the

application so that we can view what our

navigation bar currently looks like

great

okay because our navigation bar is

currently White

our brand icon is also White

so we can't currently see our graduation

cap brand

so let's change the color to Red for now

great okay

so I want to make the background for our

application as black

and address styling features that I'd

like to apply to all of our components

one way we can do this is to include the

relevant styling code within the app.css

file which is referenced by the app

component the app component is the

parent component for all our bespoke

child components so the Styles included

within the app.css file will propagate

to all the app components child

components so let's firstly import the

quicksand font from Google through this

line of code we want the quicksand font

to be applied for our entire web

application

let's use the star like this to indicate

the following CSS properties applied to

the roots tag for our single page

application

so let's first set the Border Dash

sizing property to borderbox

if you set the Box Dash sizing property

to border box on elements like this the

padding and Border values are included

in the width and height properties for

the relevant elements

let's set the font Dash family property

to quicksand

right let's create the styling for the

app element

let's set the app Root elements

background color to Black

let's set the color for our text to

White

let's set the top margin for the app

element to 10 pixels

let's launch the app

great

okay so let's create the component that

will house the registration

functionality this component will

contain a form that the user can fill

out to register the user's relevant

details with the system

including the user's name username and

password the fields on the form will be

name username password and confirm

password

we'll incorporate regular Expressions

into this dialog for the purpose of

validating each of these fields each of

these fields will be validated in real

time IE as the user enters the relevant

information into the relevant text box

controls so let's start by creating a

folder named register within the

components folder

let's add a file named register.js

within the register folder

and let's also create a file named

register.css

let's go to the register component

let's import the use statehook and the

use ref hook into our component so the

use statehook is used to track the state

of relevant variables

when the state for a particular variable

changes react detects this and

re-renders the relevant component in the

user's browser

this use ref hook will be used to

reference a text box form control

let's import relevant react bootstrap

components

let's import relevant font awesome

components

let's use rafce to generate the

boilerplate code for our register

component let's declare a constant named

name ref which will be used to reference

the name text box control

let's use the use statehook to monitor a

variable named name the set name

function will be used to change the

state of the name variable the name

variable will be bound as it were to the

relevant text box control where the user

will enter the user's name

let's include a Boolean variable for the

purpose of monitoring the valid or

invalid state of the text value entered

into the name control by the user let's

create two variables that are used for

the same functionality as the previous

two variables

that we have created

but are applied to the username text box

control so when the username is deemed

as valid the valid username variable is

set to True conversely when the username

variables value is deemed as invalid the

valid username value is set to false

let's do the same for the password or

PWD field I.E create two appropriate

variables where their state is tracked

let's do the same regarding the confirm

password field which is used to validate

if the confirm password field matches

the password field

so I've already prepared the regular

expressions for the relevant Fields off

screen

please feel free to copy the relevant

regular expressions from this location

on GitHub

so let's set a constant to the regular

expression that will be used for

validating the name field please

remember if you get stuck or you wish to

copy and paste the code rather than

following along with creating the code

line by line with me you are able to

copy the relevant code from a GitHub

repository available at this location

so this regular expression will ensure

that the name field has at least two

characters and no numbers are included

so let's include the regular expression

for the username field

the validation Criterion for this

regular expression means that the

username must be between 8 to 20

characters long

no underscore or period characters are

included at the beginning

no double underscore or underscore

period characters or period underscore

characters or dot dot characters are

included within the field and no

underscore or dot characters appear at

the end of the word

so this regular expression is used for

the password field this regular

expression means that in order for the

password field to be deemed as valid a

minimum of 8 characters must be included

the password must include at least one

letter one number and one special

character

so let's apply the use effect hook so

that a function that we'll write

that is passed as an argument to the use

effect Hook is fired when the register

component first loads so the use effect

hook accepts two arguments the first

argument is the function to be fired and

the second argument determines when the

function is fired

so we want the function passed in as the

first argument to be fired when the

register component is first loaded so we

can achieve this by passing in empty

square brackets as an argument to the

second parameter of the uzi Fair tug

so we want the name control to receive

focus when the register component first

loads we can achieve this with this line

of code

we'll write the code to create the

relevant form controls including the

name control in just a bit

and this will of course be implemented

within the jsx section of this component

let's use the use effect hook to monitor

the state of the name variable which

will be changed with every key press

that the user makes within the name text

box which as I've said will be created

soon

so to do this

we pass a function as the first argument

to the use effect hook that we want

fired when the state of the name

variable changes then in order to tell

react as it were that we only want the

relevant function to fire when the name

variable States changes we can pass the

name variable within square brackets as

the second argument to the use effect

hook

note that the test method is executed on

the relevant regular expression to

ascertain whether or not the name

entered by the user is valid or not

the result returned from this line of

code will either be true or false

we can then set the state of the valid

name Boolean variable to the result

returned from the test function

we'll create UI functionality to show

that the name field is valid or invalid

in just a bit so we are providing the

user with real-time validation feedback

as the user types the user's name into

the name field

then let's apply another use effect hook

to perform similar functionality for the

username

let's apply another use effect hook

to check that the password is valid or

invalid based on the relevant regular

expression

then we must also check whether the

password field matches the confirm

password field

note that within the array passed in as

an arguments to the second parameter of

the use effect hook

is the variable representing the

password field

and the variable representing the

confirm password field

so this means that each time the

password or confirm password Fields

change

the relevant validation functionality is

executed we'll wire up the relevant

variables to their counterpart text box

form controls in just a bit you'll see

that the on change event of the relevant

form controls is used for this purpose

right so let's create the jsx code

within the relevant section of the

register component let's create a

container component

let's create an appropriate heading

let's create a main element and a div

element within the main element

please include the relevant class name

attributes and values within the

relevant elements

let's include the react bootstrap form

element

so each text box control will reside

within a form dot group element which

will also include a label control

so let's create the relevant code for

the name text box control

the form.label element contains a span

element here

we have a ternary operator that shows or

hides either a font awesome times icon

or a tick icon based on whether the

valid name variable is true or false so

this span element will be shown to the

user based on whether the span elements

class name property references the CSS

class named valid or the CSS class named

hide we'll create these CSS classes in

just a bit

so if the valid name value is true the

fa check icon is displayed next to the

name text box control conversely if the

valid name variable is set to false this

span element is displayed to the user

which means the fa times or a cross icon

is displayed to the user indicating that

what the user has entered into the name

field is currently invalid then let's

create the text box control for the name

field like this

note that we are setting the ref

property to the name ref const this

allows us to reference the text box

within the relevant use effect hook

where we have created code so that the

name text box is given focus when the

register component first loads

then the on change event is wired up to

an arrow function that sets the tracked

name variable every time the user

changes the text entered within the name

text box

let's create the form control code for

the username field which contains

similar functionality

let's create the form control code for

the password field

and the confirm password field

let's create the code for a submit

button so we want to disable this submit

button while any of the relevant form

fields are not valid so let's set the

relevant disabled property to this

Boolean expression

so before we test the validation code

for our register form let's set up the

roots for our application

so that we can navigate to the register

component through the navigation bar

that has been created within our header

component so let's open the app.js file

let's import The Roots component and the

root component from within the react

router Dom package let's include the

roots element below the header element

we want the header elements to appear

regardless of where the user navigates

to within our application so we need the

header appropriately displayed outside

the roots element

within the roots element let's include

the relevant roots

where we can tell react rooted arm as it

were about the paths of our components

we haven't yet created the layout

component so let's do that now the

layout component provides instructions

to react as it were regarding the

overall layout of the relevant

components that are mapped to relevant

roots

let's import the layout component

we also need to create a route for the

app component and we must do this within

the index.js file

we need to wrap this route within the

browser router component as well as the

roots component like this

let's go back to the header component

and create code so that when the

register button is clicked that the

register component is appropriately

loaded

let's create a const named navigate that

is assigned a function returned from the

use navigate hook

let's create a reusable function named

handlenav

let's use the navigate function to

navigate the user to a path passed in to

the handlenev function

let's pass in the relevant path to the

register components to the handle Now

function so when the register button is

clicked the register component is loaded

Let's test the code

okay so to fix this issue we need to

correct a typo here

where we have incorrectly named the

regular expression const pertaining to

the username field

great but in order for our validation to

work as expected we need to create the

relevant CSS classes

let's make sure that we import the

register.css file into the register

component let's open the register.css

file and create the code for the

relevant CSS classes

let's create the CSS code for the

register Dash container class which will

make sure the layout for the register

dialog is appropriate

we are using the flexgrid functionality

to appropriately handle the layout let's

include the register Dash layout class

to handle the layout for the register

form

let's include the valid class so that

the elements referencing this class are

appropriately displayed

let's include the hide class so the

elements referencing this class are

appropriately hidden

let's include a class named invalid Dash

icon to display the times or cross font

awesome icon in red let's include a

class named valid icon to display the

checked icon or tick font awesome icon

in Green Let's include a class named

register Dash message

which will style a displayed message to

the user that is centrally aligned and

in white text the appropriate text will

display once a user has successfully

registered

Let's test the UI validation

functionality for the name field

Let's test the UI validation

functionality for the username field

okay

so we have an issue the name field is

also being affected

this should not be happening let's look

at the code

okay so within this use effect hook lies

the problem we are currently using the

set valid name function here whereas we

should of course be using the set valid

username function here

so let's fix this

great

Let's test the password field and the

confirm password field validation

functionality

oops and of course the password control

should be of type password and not text

we must of course also make the same fix

for the confirm password control

excellent

so let's go back to the register.css

file and make sure our register form is

responsive on smaller screen sizes

so let's use an appropriate media query

for this purpose

excellent

so now we want to interface with the

neo4j database hosted in the cloud

so we are going to download the code for

the HTTP based web API component written

in Java to our local machine so we are

going to host the web API component on

our local machines just to make the

development of our application a bit

easier

this web API component is hosted using

spring boot in order to be able to run

this code locally on a Windows computer

you must ensure that you have the Java

jdk installed on your local machine

if you don't yet have the Java jdk

installed please navigate to this URL

then follow the instructions for

installing the Java jdk Java jdk version

20 or version 17 should be fine

the next step is to clone the GitHub

repository at this URL onto your local

machine this GitHub repository contains

the Java code for the relevant web API

so we are hosting the server-side code

on our local machines we are connecting

to a neo4j database hosted in the cloud

from our local machines

please remember the local directory that

contains the cloned GitHub repository

code

we need to run a command at the command

line later on to run the relevant web

API component as stated before this will

be run locally

please ensure that you have configured

the web API component correctly to point

to the neo4j database that you have

created if you are unsure of how to do

this please follow the relevant

instructions provided in the previous

part of this course

the values I'm showing you here will of

course not apply to your particular

environment

for now let's create the client-side

react code to connect to the locally

hosted web API component

so let's create a folder within the SRC

folder named API

within the API folder let's create a

file named axiosconfig.js

let's first write code for importing

axios

let's create the code to set up the

axios client component

so let's add the base URL property

and set it to localhost port 8080 which

is where we will be able to connect to

the relevant web API endpoints this web

API component of course handles calls to

the relevant neo4j database that is

hosted in the cloud let's create

appropriate code for the headers

property

now let's go to the register component

and import the axios client component

let's create a JavaScript function named

post data

we must include the E dot prevent

default line of code for when submitting

a form from within a react component

because we'll be using the on submit

event for the submission of the register

form

then let's create an object that

encapsulates the data entered by the

user I'm going to hard code the roles

value here which is okay for demo

purposes but you'd want to handle this

field more appropriately for a

production environment for example you

could have a list of roles presented to

the user in a list box control and the

user can select the relevant role from

the list box control right so we can

write code for post requests using the

axios client component like this

so we are simply posting the object we

have just created

to an appropriate endpoint created for

handling the register post request

we can include basic error handling code

through the use of try catch code and we

can output an appropriate message to the

user regarding whether the user has been

registered successfully or not

let's include code at the bottom of the

form to display this registration status

information

let's first set the submit success

message appropriately based on the

response returned from the server-side

code

and at the bottom of the register form

let's write the code to display the

relevant message saved within the submit

success variable

so let's set the Min height property of

the app parent component so that it

extends to the bottom of the web page

and we can do this by setting the Min

height property here so that its minimum

height is 100 VH

so the result of this is that the

background color will be black

regardless of which component is loaded

because the relevant components are all

child components of the app parent

component

and of course we must wire up the on

submit event

to an appropriate event handler so we

are wiring up the on submit event to the

post data method within the form element

like this okay

so to run the web API component on your

computer

you can run this command at your command

prompt

so launch the command prompt as

administrator change the current

directory to the root directory of where

you cloned the relevant Java code

locally

just a reminder the relevant Java code

can be found at this location on GitHub

so once you have done that type this

command at your command prompt

and press enter after you press the

enter key you should see similar

information to the information displayed

on my screen logged to your screen

let's run the react code and see if we

can register a user

oops we have a bug here

we are referencing a control named PWD

with the object that we are attempting

to post

ewd field doesn't exist because we named

the relevant field password so let's fix

this

we have successfully registered a user

so we are now successfully communicating

from the react client code with the Java

server-side code which for development

purposes we are currently hosting on our

local machines

so we have successfully registered a

user the next logical step is to create

the login functionality

let's create a folder named login and

let's ensure that the first character is

lowercase within the login folder let's

create a file named login.js and let's

ensure that the first character of the

file name is an uppercase

let's create the file within the login

folder named login.css

right let's write the code for the login

component let's reference the login.css

file let's import the relevant

dependencies

let's import the use State use effect

and use ref hooks from react

let's import the container component

from the react bootstrap package

let's import the button components from

the reactor bootstrap package

let's import the form component from the

react bootstrap package

let's appropriately import the axios

client component

let's import the use navigate hook the

link component and the use location Hook

from react router Dom

let's create the boilerplate code for

our component by appropriately typing

rafce and pressing the enter key

great

let's create a const named navigate

and assign it to Value returned from the

use navigate hook

we will be able to use the function

assigned to the navigate const in code

for navigation purposes

let's assign a value returned from the

use ref hook to the user ref const

this will allow us to reference a

specific form control within a function

we'll look at this functionality in just

a bit in order to track the state of the

username let's use the you state hook

for this purpose

so user is a variable we want tracked by

react and we must use the set user

function to change the state of the user

variable

let's use the use statehook to track the

state of the entered password

let's use the use effect hook so that a

function executes when the login

component first loads

the code for this function simply sets

the username field to have a focus and

this happens when the login component is

first loaded

let's create the UI code for our login

component within the jsx section of our

component

so let's first reference the react

bootstrap container component so that

our login form is housed as it were

within an element that references the

container bootstrap Style

let's create a heading for our login

form

for styling purposes let's create a main

element and within that a div element

Within These elements our references to

appropriate CSS classes we'll create the

code for these CSS classes in just a bit

so like with the registration dialog we

create a form element and within the

form element the relevant form controls

within form.group elements these

components are imported from the react

bootstrap package and we are using them

for layout and styling purposes so each

of the form groups contain a label and a

text box

so we have the username field and the

password field within the login form

note how we are using the on change

event to change the state of relevant

variables as the user changes the text

within the relevant text box controls

let's reference the button component and

this button will serve as the login

forms submit button

please remember if you get lost at any

point while following along the full

code is available at this location on

GitHub

the link to this GitHub repository is

available in the description of this

video

let's create an arrow function to handle

the submit functionality

so let's use this code to reference the

values that the user has entered into

the username field and the password

field

let's Implement basic exception handling

through the use of try catch

functionality

basic authentication has been

implemented in the web API component

please navigate to this URL if you'd

like to learn more about basic

Authentication

so in terms of basic authentication we

must encode the username and password

values in a particular format before we

submit a particular HTTP request to the

server-side code in order to implement

basic authentication on the client we

are going to pass the username and

password in a particular format through

the header of a HTTP get request for

this particular login component the

relevant format for the username and

password is username colon password

this is essential for basic

authentication we must also encode the

username colon password value in base64

encoding

we can use javascript's btoa function

for this purpose

so let's create a const named b64 in

code and assign it the value of username

colon password

that is base64 encoded using the btoa

JavaScript function the next step is to

create an object representing the Header

information that will be passed to the

relevant web API endpoint

so let's create a const named config and

set it to an object to represent

appropriate HTTP Header information this

object contains a property named headers

which is set to an object

with a property named authorization the

authorization property is set to the

text value basic followed by a space

followed by the base 64 encoded username

colon password text value

note we are wrapping the value assigned

to the authorization property in

backtick characters

so that we can interpolate the relevant

text value this means we can include a

variable within the baptic characters by

wrapping the variable within curly

brackets and preceding the curly

brackets with the dollar symbol

let's create a request to the relevant

endpoint using axios note we are passing

in the username and password through a

HTTP get request made to the relevant

endpoint through the config object

the config object contains the

authentication details for the user

which is appropriately added to the

header of the relevant HTTP get request

so before we test the login

functionality let's make sure that an

appropriate route is set up for the

login component

so let's open the app.js file and

include the code for the relevant root

let's import the login component into

the app component

and include code that represents the

route for the login component

let's open the header.js file and

include navigation functionality in this

component so that the user can navigate

to the login component by clicking the

login button displayed on the navigation

bar

Let's test the code

we have a few issues here so let's fix

these issues

please see the relevant GitHub code at

this location if you get stuck on any

issues

here this should be used State and not

use ref

there are also a few other issues that

need to be fixed so let's fix these

issues

let's run the code

so we have our login form presented to

the user but it doesn't look great we'll

fix this in just a bit so off screen I

registered a user named Henry Jamison

please feel free to register your own

users while testing

I'm going to test the login

functionality using Henry Jamison's

credentials

I'm going to use a simple JavaScript

alert to ensure that the HTTP status

sent back from the relevant endpoint is

200 meaning that the HTTP request has

been successfully processed on the

server

excellent we have received a response

with a status code of 200 so we've

received a HTTP response with a status

code of 200 meaning success

so next let's style the login screen

appropriately

let's open the login.css file let's

style the CSS class named login Dash

container

we are using the flex grid for styling

and layout purposes the flex grid is

great for creating responsive UI code

for various screen sizes let's create

the login Dash layout CSS class

we are also using the flexgrid

functionality here note that we are

setting the flex Direction property to

column which means the relevant elements

will stack one on top of the other in a

column layout by default if the flex

Direction property is not set this

setting defaults to row this means the

relevant elements are presented one next

to the other in a row layout

let's give each of these form.group

elements suitable margin bottom settings

using bootstrap

so that we are creating appropriate

spacing between the form groups

so that our layout here is a bit neater

great

let's include an error message at the

bottom of our form for if something goes

wrong during the authentication process

so let's set the state of an appropriate

error message based on the response

provided by the server

so if a status code of 200 is returned

from the server the state of the ER

message variable

can be set to an empty string or if the

status code does not indicate success or

an error is thrown we can set the ER

message variable appropriately to

indicate that the login has failed on

the server if an exception is thrown we

can include the error message that has

been caught within the try catch block

then let's output the value stored in

the ER message variable to the screen at

the bottom of the login form

let's style the error message display

appropriately

so to test this I'm going to force a 404

exception meaning that the relevant

endpoint is not found

by changing the endpoint path to

something that isn't correct

great

lastly let's ensure that our login form

looks good on smaller screens so to do

this within the login.css file let's

include appropriate media query

functionality

let's include code here to automatically

navigate the user to the home page once

the user has successfully logged in

excellent great so once the user has

been authenticated we want our other

components to know as it were that the

user is logged into the system so the

best way to do this is through the use

of context

react context is a way to manage State

globally so very basically we use react

context so that other components will be

able to retrieve a value

from a global store and assess whether a

user is logged into the system or not

logged into the system

for more information about react context

please navigate to this URL

so in order to create context let's

create a folder named context within the

SRC folder

within the context folder let's create a

file named auth provider.js

let's import the create context function

and the use statehook from react

let's set a const named auth context to

a value returned from the create context

method like this

note that we are passing an empty object

to the create context function let's

write code to export a component named

auth provider note that this code simply

means that the auth variable and set

auth function will be passed down to all

components that are wrapped by the auth

provider component so this makes these

accessible to the relevant child

components that are wrapped by the auth

provider parent component we'll look at

this in just a bit so what this means is

that all relevant components will be

able to access the auth variable with an

all relevant child components if the

auth variable is not null this means

that the relevant user has been

authenticated I.E logged onto the system

of course the state of the auth variable

is managed through the use of the use

statehook implemented within the auth

provider component

let's create a folder named hooks within

the SRC folder within the hooks folder

let's create a hook named use auth this

code basically allows relevant

components to access the auth context

which means access is provided to the

auth variable and the set off function

the set or function of course is used

for changing the state of the auth

variable so to log in you change the

state of the auth variable and to log

out you appropriately change the state

of the auth variable by setting it to

null the auth variable State indicates

whether the user is logged into the

system and the set auth function can be

used to change the state of the auth

variable

so the set or function can essentially

be used to log the user into the system

or out of the system let's create a hook

named use axios private

so the code for the hook is created so

that the client code can retrieve an

appropriate axios client object that

automatically includes an appropriate

HTTP header containing the logged on

user's credentials these credentials are

appropriately formatted for basic

Authentication so The Returned axios

client object from this hook must be

used for accessing protected resources

on the server I.E retrieved through the

relevant API endpoints these protected

resources require basic authentication

to be performed before returning

relevant data to the client

now the code here creates what's known

as an Interceptor so if they use axios

private Hook is used from within client

code to reference the axios client the

login credentials are automatically

injected into the header of the relevant

HTTP request so this means that the

client code doesn't need to inject this

authentication Header information every

time a HTTP request is made to an end

point this Interceptor code does this

automatically just a reminder if you get

stuck creating this code please

reference the relevant code at this

GitHub repository location

so the next step is to open the index.js

file and wrap all the relevant

components within the auth provider

parent component as discussed this

allows the child components or makes

Provisions for the child component to

easily access the auth variable so that

the relevant component can access

whether the user is logged into the

system or not logged into the system

so let's look at how this works let's

open the home.js file let's import the

use auth hook

let's use the use auth hook

to return the auth variable and the set

or function from the relevant context

Global store

now within the UI jsx code section of

the home component we can check the auth

variable to see if the user is logged

into the system like this

so if the user is logged into the system

the code displays a message on the

screen you are logged in if the user is

not logged into the system the code

displays the hero image to the screen so

let's open the login.js file and use the

use auth hook to return the set or

function from the context

let's use the set auth function to

assign an appropriate object to the auth

variable this object contains the

username and password

so once the user is successfully

authenticated

an object containing the user's username

and password

is passed into the set or function which

effectively changes the state of the

auth variable

which indicates of course that the user

is logged into the system

so you can see that now this Global

context can be used to retrieve the

username and password from this object

I.E the object that we have just passed

in to the set auth function to change

the state of the auth variable

the username and password can now be

injected into the header of HTTP

requests through the Interceptor

functionality that we have implemented

in the use axios private hook

let's run the code

let's register a new user named Brandon

Lowe

okay the password Here is password 1

exclamation mark

let's then log into the system

excellent

so let's create our logout functionality

within the header component

let's retrieve the auth variable and the

set auth function from the use auth hook

so if the auth.user property is not null

this means that the user is logged into

the system

so we want a log out button to be

displayed in the navigation bar rather

than the login and registration buttons

when a user is logged into the system we

only want the login and registration

buttons presented to the user when the

user is not logged into the system let's

create a function for when the logout

button is clicked

so let's create this logout function

within this function we can simply set

the auth variable to null by using the

set auth function for this purpose this

effectively means that the user is now

logged out of the system

then let's also Implement code to

navigate the user to the home component

right let's log into the system

great

let's log out of the system

excellent

so let's create the code within the home

component so that when the user is

authenticated a list of courses is

displayed to the user rather than the

hero image

let's create code using the use

statehook

so that react monitors the state of a

variable named course data we can change

the state of the course data variable

using the set course data function

returned from the use statehook

let's use the use axios private hook to

return a reference to the axios client

that we are going to use to request a

list of courses from the server

note we are getting a reference to the

axios client using the use axios private

hook because the use axios private hook

contains functionality that will

automatically inject the logged on

user's credentials into the header of

the relevant HTTP request so each

request to the server requires the

user's credentials so that basic

authentication can be appropriately

performed

this is because the course data is a

protected resource so the user must be

authenticated before this course

information is passed from server to

client remember implemented within the

use axios privatehook is Interceptor

functionality so the HTTP request is

intercepted and the relevant

authentication information is injected

into the header of the HTTP request but

for the relevant HTTP request is made to

the server

so let's create a JavaScript Arrow

function named fetch courses so here a

get request is made

and once the results are returned we are

using the set course data function to

appropriately change the state of the

course data variable this results in the

home component re-rendering which means

a list of courses will be displayed on

the UI once the data becomes available

and the set course data function is

appropriately called to change the state

of the course data variable we'll

implement the code for displaying the

course data on the UI in just a bit

let's use the use a fat hook to call the

fetch data function when the home

component first loads but we only want

the fetch data function to be called if

the user is logged into the system

let's display the course data value in

an alert to test our HTTP get request

let's run the code

oops we currently are not importing the

use statehook let's fix this

we now have a 404 error this is

occurring because we need to include a

forward slash at the end of the endpoint

path here

let's try again

and we are getting the expected Json

results with a now alert this means our

HTTP request is working as expected

right so let's write the code for

displaying the course data on the UI

so in the return section of the home

component let's implement the code for

displaying the course data

I'm going to create this code fairly

quickly with minimal explanation

so we are using the map function here to

iterate through the courses list

returned from the server this courses

list is stored within the course data

variable

so I'm deliberately not providing a lot

of explanation here please note that

this part of the course is not a

detailed guide on bootstrap or even

react for that matter this part of the

course has been created for the purpose

of placing the advantages provided

through the use of the neo4j graph

database management system in the

context of a web-based application

please feel free to copy the relevant

code from GitHub the first map function

let's refer to this as the parent map

function traverses the list of courses

note that we are outputting the title

for each course within the parent map

function this title property

will be returned from the relevant

endpoint

within the parent map function another

map function is implemented that

traverses a list of lessons the lessons

make up the relevant course

so we are outputting the title of the

course as a heading a list of lessons is

outputted for each of the relevant

courses underneath the relevant heading

so essentially the data is grouped by

course

the title for each course and the

course's lessons are outputted to the UI

bootstrap is being used for layout and

styling purposes

note that we are setting the key

property for this element to the

identifier property for each course

when using the map function it is always

a good idea to appropriately set the

react key property for the relevant

element like this this helps react to be

more efficient when considering the

component

we are also appropriately setting the

key property for the element here

related to each lesson we are setting

the key property here to the identifier

of each lesson

Let's test the code

okay we have an issue here so to fix

this issue we need to import the link

component from the react router Dom

package

we are using the link component to wrap

the title for each course

this is because we are going to create a

component responsible for displaying

course details for each individual

course in a bit

once this is done we'll point this link

to the relevant course component where

the details for the relevant individual

course will be outputted

this will allow a user to click the

header for a course and for the course

details for the relevant course to be

displayed after the user clicks the

relevant link on the home page

you can see we have four courses

currently saved to the system

so let's create a component that

displays the course details for a

particular course

so

from a ux user experience perspective

the user will click on the course

heading on the home page and will be

navigated to a component that displays

the course details for the relevant

course

let's create a folder named course

and a file within the course folder

named course.js

let's generate the boilerplate code for

the course component

let's open the app.js file and include

an appropriate route for our course

component

note that this component accepts a

parameter passed into the course

component

which is an identifier for a particular

course

so to tell the react router Dom system

as it were about the relevant identifier

parameter we can include a colon

followed by the name of the parameter

identifier at the end of the relevant

root path like this

let's go back to the course.js file

let's include the relevant Imports

I'm not going to explain this import

code as we have already covered this

while creating code for the other

components

all except for the use params hook

we are going to use the use params hook

to extract a parameter value that will

be passed into this component from the

home component when a user clicks on a

course heading

so let's create a value from the use

params hook

we can then use the params object to

retrieve the name of the parameter that

we named identifier

let's create a tracked variable named

course data we can use the set course

data function to change the state of the

course data variable

of course when the state of the variable

returned from the u-state hook has

changed this results in the relevant

component being re-rendered

let's use the use axios private hook

to get a reference to the axios client

where an Interceptor is employed to

inject the user's authentication details

into the HTTP header for any HTTP

request made using the axios client

returned from the use axios private hook

let's use the use effect talk so that we

can execute code that retrieves data for

the relevant course when the course

component first loads

let's use an alert to Output the results

returned from the relevant get request

within the home.js file let's include

the appropriate path to the course

component within the relevant link

component like this

note we are including the course

identifier as part of the path assigned

to the two property of the link element

Let's test the code

great the results are outputted through

a JavaScript alert the JavaScript alert

is of course just used for testing

purposes

let's create the UI code in the part of

the course component that returns jsx

code

so the code is very similar to the code

within the jsx section of the home

component so to speed things up I'm just

going to copy and paste the relevant

code from the home component

into the course component then let's

adapt the code appropriately for the

course component

so the difference between the relevant

code and the home component and the

relevant code in the course component is

that only one specific course is

displayed in the course component

whereas a list of all courses are

outputted from the home component

just a reminder if you get lost while

creating a code for any part of this

application please reference the code

that can be found at this location on

GitHub

note that for our course component we

are also including the instructor that

teaches the relevant course

this data is returned from the server in

the teacher property

so the course component only ever

returns one course rather than a list of

courses that are displayed within the

home component

so we copied this code from the home

component which is currently incorrect

for the course component we need to

replace the D variable here with course

data because the D references an item in

a list of courses in the home component

this course data variable represents one

particular course

excellent

the next component that I want to create

is used for the purpose of displaying

only the list of courses in which the

logged on user is enrolled

so let's create a folder within the

components folder named enrolled courses

within the enrolled courses folder let's

create a JavaScript file named enrolled

courses.js

let's copy and paste the relevant

imports from the home component

so for this component I'm going to copy

most of the code from the home component

as the code is very similar

the only difference is that we are

calling a different endpoint on the

server to get the relevant course data

in this case only a list of courses in

which the logged on user is enrolled

will be displayed to the user

just a reminder you can of course get

the finished code from the relevant

GitHub repository

let's include an appropriate root

component for the enrolled courses

component within the app.js file like

this

now let's open the header.js file and

include code so that a link to the home

component and the enrolled courses

component will appear within the

navigation bar at the top of the UI

in order for our links to be displayed

in white text let's include the

following properties within the navbar

element like this

so on smaller screens this navigation

bar doesn't currently look very good

so let's wrap the relevant navigation

elements within the nav collapse element

so that on smaller screens the

navigation bar responds appropriately

great

we copied this code from the home

component and we of course don't need

this code that outputs the hero image in

the enrolled courses component so let's

delete this code here

currently when we click on the enrolled

courses link no data is displayed

and this is an expected result because

the logged on user has not enrolled in

any courses

so let's create the functionality so

that a user can enroll in courses

we'll implement this functionality from

within the course component

so let's create code that displays a

button within the course component that

is labeled enroll

this button must only display if the

enrolled property for a particular

course object returns false

which denotes that the logged on user is

not currently the enrolled in the

relevant course

so when this button is clicked we want a

function named enroll to be executed the

code for this enroll function will of

course enroll the logged on user into

the relevant course

so let's create the enroll function

you can see that we are using a HTTP

post request

to post the course identifier wrapped in

an object to a specific end point the

server-side functionality written in

Java interacts with the neo4j database

management system appropriately in order

to create the appropriate relationship

between the user and the relevant course

so if the endpoint returns a status

response of 200 meaning success let's

display this message to the user which

simply thanks the user for enrolling in

the chosen course

then let's include code to navigate the

user to the enrolled courses component

Let's test this functionality

so each of the lessons listed for the

enrolled courses must have a play button

icon next to it when the user clicks the

play button icon the lesson that has

been created as a YouTube video will be

displayed to the user through the video

component that we are about to create

so the play button is implemented using

the font awesome icon component

the font awesome icon element is wrapped

by the link element the link elements to

property is set to the path of the video

component that we will create

the lessons identifier is passed in as a

parameter to the video component when

the relevant video link is clicked the

play button icon denotes this link on

the front end

so in the home component we are

currently handling what to Output to the

screen the hero image is displayed when

the user is not logged into the system

and a list of courses is displayed when

the user is logged into the system when

the user clicks the enrolled courses

link this is currently not being handled

appropriately

we want the login screen presented to

the user if the enrolled courses link is

clicked while the user is not logged

into the system

then when the user logs into the system

we want the user to be navigated to the

enrolled courses component rather than

the home component so let's implement

the code to handle this scenario

let's first create a component within

the components folder named required

auth

this component is responsible for

navigating the user to the login screen

if auth is null I.E the user is not

logged into the system

then let's open the app.js file and wrap

the enrolled courses root

with the required auth element like this

so the user will not be able to access

the enrolled courses component unless

the user is logged into the system if

the user tries to click the relevant

link the user will be presented with the

login screen so this forces the user to

log in in order to see the user's

enrolled courses and once the user logs

in the user is navigated directly to the

enrolled courses component

which means the user can see the list of

courses in which the user has been

enrolled within the login component we

need to create the navigation code

so that when the user logs in the user

is navigated to the component that the

user tried to access prior to being

redirected to the login screen in this

particular case this would be the

enrolled courses component

Let's test this code

great

so the last component that I want to

create is of course the video component

this component uses the react player

component to play the lessons that have

been created as videos on YouTube to the

user let's create a folder named video

let's create a file within this folder

named video.js

and let's create another file within the

video folder named video.css

the code for implementing the play video

functionality is very basic

let's use the use param hook

so that we can retrieve the YouTube

video ID that will be passed into the

component like this

let's open the app.js file and include

the appropriate route

so that the react router Dom system as

it were knows how to navigate the user

to the video component as well as pass

in the appropriate parameter to the

video component the parameter value will

be the YouTube video ID pertaining to

the relevant lesson

let's include the react player element

appropriately within the video component

so that the relevant YouTube video plays

immediately

we are setting the playing property to

true

for the react player element for this

purpose

then let's set the URL property

appropriately to point to the relevant

video on YouTube note that the URL

includes the YouTube ID that is passed

in as a parameter to this component

let's style the video component

appropriately

right let's test the code

I'd like to introduce you to this

exciting new Unity for beginners course

where we'll create a game using unity

and c-sharp over a series of videos

all the software that we'll use

including unity and vs code can be

downloaded free of charge

this course is totally free

hi and welcome I'm Gavin long the rigid

body component allows us to apply

physics to a game object for example

gravity but we are going to write code

in the update method to apply a forward

Force to the player game object in

response to the user pressing down the W

key on the user's keyboard

so to improve the user's experience you

can include a spinner that appears to

the user while data is being loaded so

in fact the last component that we'll

create could be the spinner component

so when the data is loading we present

the user with a spinner and then when

the relevant data becomes available the

spinner component disappears and the

data is displayed to the user

so for this purpose I've created a

spinner component

please reference the relevant code on

GitHub if you'd like to include the

spinner functionality within your

application

I'm going to fast forward the creation

of the spinner component as this is

purely a bonus piece of code that you

can study at your own pace if you are

interested to implement the spinner we

are simply creating a react component

relevant CSS code to style and animate

the spinner

and functionality within the relevant

components to appropriately show and

hide the spinner the showing and hiding

of the spinner is implemented using a

Boolean variable

where the state of the relevant variable

is tracked through the use of the use

statehook from within the relevant

components showing or hiding the spinner

please review the code in detail at the

relevant GitHub repository location

the link to the relevant repositories

available Below in the description of

this video

we can pass in the relevant Boolean

value as a prop to the spinner component

to indicate to the spinner component to

be visible or hidden like this

let's integrate the spinner component

into the enrolled courses component

hi and welcome I'm Gavin long

this is part two in a course where we'll

create a game using C sharp and unity in

part one I demonstrated the

potentially making the game at runtime

more robust for details on c-sharp data

types I've included a link to a c-sharp

tutorial on C sharp date

so I'm going to finish the code for this

application by styling the header of the

courses displayed in the home component

the course component and the enrolled

courses component

I'm including the relevant CSS classes

within the app.css file so that the

classes affect all of these components

I know the UI for this application is

not beautiful but the purpose of this

part of the course is to place an

application that takes advantage of a

neo4j graph database management system

for the purpose of managing

relationships between data entities

in the context of a web-based

application

my challenge to you is to Fork the

GitHub repository that contains the code

that I've demonstrated in this part of

the course and Abstract the common parts

of the jsx code contained in the home

component the course component and the

enrolled courses component into relevant

child components

one suggestion is to create a reusable

child component for an individual course

you can then use appropriate props to

dynamically customize the new child

course component based on its context

I.E its parent component for example in

the enrolled courses component a play

button appears next to every lesson for

each course

whereas in the home component these play

buttons are not included next to any of

the lessons for each course so the new

course child component will need to

account for these two scenarios and

dynamically output the appropriate jsx

code

I hope that you have enjoyed this part

of the course and have learned a little

bit about creating a client react

application that interfaces with a web

API component through axios where basic

authentication is employed thank you and

take care