(58) Prisma ORM Full Course 2025 | Become a Prisma Pro in 2.5 Hours – YouTube

By PedroTech

https://www.youtube.com/watch?v=gimSKEsWYb4

Transcript:

(00:00) Welcome to my Prisma crash course for beginners. In this video, you're going to go from a complete beginner to mastering Prisma or building a full stack job posting app from scratch using Nex.js, Postgress, SQL, and Prisma. This course is designed for beginners. Whether you've never used a database or ARM before or you're looking to level up your back-end skills, you'll learn how to define schemas, query your Postgress SQL database with type safety, and connect everything into a realworld application. By the end of the video, you'll be able to build a website with

(00:35) Prisma where users can browse different job opportunities, post a position online, apply for a job, and even see the status of the jobs they've applied for. If you want to check the code, everything will be in the description. And in order to learn complex technologies like Prisma, you need to have hands-on experience.

(00:56) And that's where today's sponsor, Brilliant, comes in. When I'm building a full stack app like the one we're going to be building in this video, it's not just about writing code. It's about solving problems and thinking clearly about how everything connects. And that's where Brilliant comes in.

(01:13) Brilliant is a learning platform where you learn by doing. With thousands of interactive lessons in math, data analysis, programming, and AI, you'll be able to improve your skills and progress in your career. What I love about Brilliant is that it doesn't just throw lectures at you. Every lesson is hands-on. You solve problems, get feedback, and build intuition.

(01:32) It's six times more effective than watching passive videos. It's also great for developing real problem-solving skills. Their algorithms and data structures course, for example, helped me sharpen the exact thinking I use when designing database queries, structuring models, or writing efficient back-end logic.

(01:52) And because lessons are bite-sized, it's easy to make learning a daily habit, even if you only have 10 minutes. So, to try Brilliant for free for 30 days, head to brilliant.org/pedrotech or click the link in the description. If you click that link, you'll also get 20% off their annual premium subscription. One easy way to support the channel is checking out our sponsors.

(02:11) So, go ahead and check out Brilliant. And now, let's get into the video. So, let's start the video just talking a little bit about what is Prisma and why it is useful. So, first of all, Prisma is a very powerful orm also known as an object relational mapping tool. So it is primarily used for JavaScript and TypeScript and it allows you to simplify the way you interact with a database by providing an intuitive and type-S safe API.

(02:40) Now what exactly does that mean? Well, an ORM is a tool that helps you work with a database using regular code instead of writing SQL manually. If you've ever written SQL in the past, you know how sometimes it can get complex and overwhelming. So, Prisma and other arms allow you to simplify that by using JavaScript functions instead.

(03:04) So, for example, if you were to want to make the following query on the screen where you select data from a user's table, instead of writing that SQL statement, you could then use the Prisma API to find that exact data and retrieve it in your app. So, why exactly is that useful? Well, there's a variety of reasons why you might want to use an ARM.

(03:24) Uh first of all, it's a lot easier to read and write data because you use the same language you're coding in without having to learn SQL uh when you're writing your project in Typescript. Also, it is safer because it prevents mistakes that could lead to breaches like SQL injections. Also, it's faster to develop because you don't have to keep writing repetitive SQL code because uh you can just use a function for that.

(03:53) And on top of that, there are clear models that Prisma allows you to define in your code, which makes it such that your entire codebase is type safe whilst adapting to how your database is structured. So that's all benefits that any ORM has. But why Prisma is specifically the one that I think is the best? Well, Prisma is extremely good compared to other arms in three primary areas.

(04:18) Type safety, performance, and community. And that's why in this video, we're going to be learning Prisma. Starting off by just building our first queries and setting up our project and then actually building a full stack project using Prisma. Okay, everyone. The way we're going to start this is I have this starter project that you can access by clicking the link in the description. This is a simple um Node.js Express server that is running.

(04:42) We're going to start the tutorial just explaining Prisma in this very simple environment so that we can then transition into explaining Prisma in the context of using a framework like Nex.js and later on even building a project with it. So if you open up this starter all that we have here is a simple project set up without Prisma yet. It just has simple TypeScript node one express all that kind of stuff.

(05:06) And there's an index.ts s over here which just generates a simple express node server. So if I were to run over here npm rundev we would run our server and we'll be having access to all the endpoints that we decide to create. Now in this project if or in any project that we want to install Prisma we can use the Prisma CLI.

(05:32) So what I want to do in my project is I want to install a couple packages. So the first one is going to be npm install Prisma and this is just a generic Prisma library which is going to allow us to create our Prisma databases which is going to allow us to actually access Prisma as an ARM. But we also have to install at Prisma/Client which basically allows us to access and interact with Prisma in our JavaScript or TypeScript project. So, I'm going to install both of them.

(06:02) And when they're done installing, we can run the command npx prisma init. And it will generate a lot of the files that we need to actually have Prisma set up in our project. When this is done running, I'll be back. Perfect. So, as you can see over here, we have finished generating and our Prisma schema was created.

(06:26) It lists out some of the next steps that we might want to follow, but I'll explain to you guys what we need to do. So the first thing that we get here is this Prisma folder and this is what is going to include your Prisma schema. So when you click on this you should see a file like this. Now if you don't get you know the syntax highlighting and so on like I do it might be because you need to install the Prisma extension.

(06:49) So you just go over here you search Prisma and you install this package over here. It is from prisma.io io and then you can have um syntax highlighting on files that are of the extension.prisma. Now this file is extremely important because as the name implies this is your schema. Not only you will define how the schema of your database tables is going to look like but also here's where you provide what database you're going to be hosting.

(07:15) So you you can actually use Prisma for a variety of different types of databases. Now in this video we're going to be using Postgress as the provider for our database. But you could use anything you want to be honest. So we're going to keep it as Postgress over here.

(07:33) But if you're using another database management system, you would replace the provider value over here with whatever you're using. Also you see we have to specify what database URL we are uh going to be using. So where are we hosting our database? For some people if you're using especially if you're using something like posgress you might be hosting it locally but I think that you should be hosting it um in the cloud especially if you're going to be building a project that's how you usually do it.

(08:00) So in this video we're just going to be using neon which is just one of the free options for um providing serverless Postgress databases. So I'm going to show you guys real quick just to set how to set up your free Neon database and I'll be back in a second. So, you should go to either the link in the description or you can just go to neon.

(08:19) com and you click sign up and then create your account. You can log in with GitHub, with Google, whatever you want. I'll log into my account and when I'm done, I'll be back to show you guys. So, when you are creating your account, you'll be prompted with this, which is a place for you to create your first project.

(08:37) Now, uh you need to give a name to your project. I'm just going to call it uh Prisma course but you obviously call it whatever you want and you can choose the Postgress version that you're going to be using. Again uh Neon is specific for creating serverless Postgress databases. So uh that's why it is mainly tailored for Postgress databases. So then you need to choose what your service provider is going to be.

(09:00) I'm going to choose AWS just because I I work at Amazon so I prefer AWS. Uh then I'm going to choose where. So, I'm going to choose here the west cuz it's closer to me, but choose whatever region you want. Uh, that it's probably the better you choose the one that is closest to where your users are going to be at. And we're going to click create project.

(09:22) Then, as you can see, we have this dashboard over here. But the important part is that we can now connect our database to whatever we want. We have here the link to our database that we can just copy and just insert it into our URL provider. So we could just put it directly over here. But as you can see this is wrapped around by an env uh function which means that this will be treated as an environment variable.

(09:52) So you shouldn't be putting your uh database URL directly into this. You should actually put it into av file which we have it over here. Now, this is not the database that we we have. This was autogenerated by the Prisma uh library when we ran Prisma.init or Prisma init. Uh but we're going to delete this and we're going to paste the database that we now have hosting on Neon.

(10:16) And we should successfully set up our database for us to use. Perfect. Now, we are ready to start building our first Prisma schema. So a Prisma schema is just a file where we're going to define our database models, relationships, and configuration using the Prisma syntax. So there's a couple important keywords that you need to learn for when you are dealing with the schema in Prisma. The first one is the generator.

(10:41) This basically just tells Prisma to generate the client code you're going to be using on your app. So, if we didn't have this over here and we didn't have the Prisma client installed, we would have a Prisma uh database, but we wouldn't actually be generating type- safe values for us to use in our project. So, we do have to have it like this.

(11:04) It would generate an output of our Prisma schema into whatever path we pass over here so that we can then access that in our project. Now, the second one is a data source. And a data source is basically us telling Prisma which database we're going to connect to is what we just spend time looking at. Now the next is a model. And the model is what we're going to be using to define our database tables and their fields.

(11:23) So as an example here for this video, we're going to start off just building the simplest model that I can think of for a project. It's a model that I think every project has, which is the model for a user. So to define a model you just write model and then you give the name for the model or for the table. So we're going to call it user.

(11:44) Then you have to define the fields that this model is going to have. So let's think about what fields at least let's start very basic. What is the most basic field that a user has to have? Well, in reality, especially if you're working with SQL databases, uh you should be having an ID in every table. So the most basic field is going to be an ID field.

(12:06) So in order to define an ID field in Prisma, I write the name ID. Then I put some space and I define what is the type of this field. And there's a variety of different Prisma types, but for an ID, it is an int. Then we have to define that this is a specific ID. So for ids specifically we can actually put this cool decorator called the at ID which will just mark this field as the primary key for this table so that uh every row needs a unique ID when we are creating this table.

(12:45) Now we can also add this other marker which is a marker to tell Prisma to whenever we create a new row in this table it is actually going to auto increment this uh field. So we can start the ID with one with two with three whatever we want but we'll let's imagine with one and if we add the decorator at default and then we put auto increment this will basically tell Prisma to automatically increase the number for every time we add a new row.

(13:17) So this is just the most basic example of how you would build an ID in every single table that you create. So you would be using this in every table that you make. Now this default that creator over here even though we just used here to auto increment it actually can be used in other fields as well.

(13:35) For example what is a field that is also very crucial for a user? Well the name field right the name is a is a thing that every user has to have. Now with name we have to think of what the type is and a type for name is string. Now a string a name is not an ID but if we wanted which I don't think you should we could put a default value to it as well.

(13:55) If we came over here and we put default and we put I don't know Pedro. This will just make it such that this field in the table will always default to Pedro uh if nothing is added. Uh you can use this for a lot of different things. For example, you can use this. very useful for when you're making a created at field cuz then you can just pass the the you can like if I were to make here created at. I could make it into a date time and then pass a default value as now.

(14:26) And that's another function that you can use in Prisma to get the current time. Now we'll delete this. We don't want to also put a default value for Pedro. It was just as an example. Now another field that we might want to be able to add is an an email field.

(14:44) So an email would also be a string but differently from name an email would have to be unique. So we can add this unique property over here to it which will basically prohibit us from adding two users with the same email. And just like that we made the most basic model that we can think of. Now we just wrote code for it right? We didn't actually make it on our database.

(15:08) If you go here to uh the tables, we don't see any tables in our neon database. So we can actually generate publicly our table here directly from Prisma by just creating the models and Prisma will do the work for us. And to do that, we have to migrate our Prisma schema. So I'm going to open up a terminal over here.

(15:30) And in the same folder that I'm in, I'm going to run npx prisma migrate. And then I'm going to put dev and d-name. And I'm going to give it as init. Now, let me break down what this means. Well, the npx prisma will obviously get us to be able to execute Prisma commands. Then the migrate command is basically telling Prisma to apply the schema changes to our database.

(15:57) Then the dev over here is just to say to Prisma that this is all in development. If we want to actually push this in production, you'll see later we have a different command for that. And then we can give a name for this migration. So every time you make changes to your tables in uh your posgress or whatever database you're using, it is good for you to keep track of all the different migrations that you did. So you give a name and since this is the first time we're making any changes to our table, I'm going to call it init. So we do d-name and then the

(16:25) name of the migration. When I run this, it will take a bit, but if everything that we've done so far was set up correctly and the environment variables were correctly loaded from ourv file, we should see that the migration was executed perfectly. So first of all, let's see what happens here.

(16:44) It generated this migrations folder which includes a record of all the migrations that you've made with your table. all the changes you've made of your table, which you might not be thinking is that big of a deal, but when you're building a large scale project, this is important for recordeping.

(17:03) But most importantly, when you go to Neon and you refresh the page, you should now see the table that we just created, exactly how we defined it. So, we see here a table with the ID field, the name field, and the email field. Not only that, but Prisma also gives to us a migrations table with a bunch of stuff that they they set up themselves. So, uh we don't even have to worry about keeping track of that. It's on the project, but it's also on our um hosted database.

(17:29) So, this is how we set up our first table using Prisma. Now, how do I actually access uh and make changes to this table using Prisma? Because like I explained in the beginning, Prisma is an ARM, right? It's supposed to make it easier for us to interact with this table and with our databases uh by using a connection with TypeScript or JavaScript.

(17:55) So the way we do that is we need to generate the code for that our client can access our schema. And to do that, here's the second important command that you need to know. The Prisma migrate was the first one that you need to definitely know. So that's when you make changes to your table. But here is what you need to know for how do you generate those changes and have access to it in your client.

(18:14) To do that, you run npx prisma generate. And when I run that, you should see that now we have generated the prisma client. And whenever we want to access it um in our code, we will be able to. So let's try it. So we don't have actually any data right in our table. We could manually just add the records here and fake some data.

(18:39) But I want to actually create this what this thing which is known as a seed. Basically, we're going to populate our table with some fake data, some initial data just for us to test. And I'm going to make here a file in our Prisma folder called a seed.ts. Now, this is not that important of a topic of a concept. is just us adding some uh initial data to our table and it just the name is seed because we're seeding the data right we're seeding we're seeding the table by giving it data so I'm going to make here an async function called seed which we're going to call uh to actually

(19:14) implement and send that data now in this function we need to access Prisma so I'm going to import at the top over here from at Prisma Prisma client and I'm going to import the Prisma client class. So let's do Prisma client. Then I need to create an instance of this class. So I'm going to say const Prisma is equal to new Prisma client.

(19:41) Now with this Prisma variable over here, we have access to altering and executing commands to our table and it has full detail of how our schema is because we ran the generate function. So for example, if I want to access and create um data on the user table, which is the only table we have.

(20:06) But before we do this, uh I actually we have to make one change. So in our schema.prisma over here, we actually have to delete this line, the output line. We want to delete this and also delete the generated folder. Uh I forgot to do this in the beginning or else our type definitions won't work. So now that we made some changes I actually to our schema I actually want to run the npx prisma generate again.

(20:31) And now what we should see is that I can actually call our Prisma model by first of all saying await because this is an asynchronous execution. then calling the Prisma client and then when I click dot you should see that we now have a list of our models specifically the user model over here. When we click on that we actually have a bunch of things that we can do with it.

(20:56) Uh these are all functions that correspond to a S SQL statement that you could then run uh in your actual table. But instead of doing that we're going to do it directly using uh Prisma as the or So in a seating function what we want to do is obviously create and populate our database with initial data just for testing.

(21:16) So we want to create data. So we want to add stuff to this table. So if you want to add stuff to a table, you can call the create function. And this function allows you to pass information about the data you want to add. But in our case over here, we want to add more than just one thing to the table.

(21:39) So I'm actually going to be calling the create menu function which allows you to do the same thing add uh some data to a table but add multiple things to a table. So the way this works is very simple. You just pass an object over here with a data property and it allows you to pass an array and in this array instead of passing data in the format that a SQL table might expect we're gonna actually pass it as a JavaScript or TypeScript object.

(22:03) So for example, if you look at our model, it's very simple, right? Uh there are three fields, but two of the fields are very similar and one is very different. The ID field, we don't have to pass anything for it because it autogenerates and auto increments as you can see.

(22:20) So we only have to pass data for the name and for the email whilst following uh following its type definition. So let's start with the name. I'm going to pass here name which already has autogenerated types because like I said Prisma knows what the type is. And we're going to pass here for example the name Alice. Now we also have to pass an email.

(22:40) This email is going to be Aliceample.com. Now let's add another user just to populate this. I'm going to copy the same format and we're going to add the second user. It's going to be Bob and we're going to put bob atample.com. Now we have this seed function that when we call is going to hopefully execute the create command on Prisma and add two different entries to this table.

(23:10) So how do we call it? Well, we can call it down here by saying seed that we we just call the table the the the the function. But it's also important that we disconnect the connection between our project with our Prisma client. So one thing we can do as well is after this is done, we can run this uh Prisma.connect.

(23:36) It's a way for us to prevent any leaks from happening uh due to us not closing the connection. Now, in order to run this seating that we created over here, we have to go to our package.json and we're going to create a Prisma key uh inside of here. We're going to say Prisma and we're going to add a key inside of the Prisma key called seed.

(24:08) Now, in here, we're going to run the following command ts node and then put the path towards our seed file. So it's going to be prisma/seed.ts. So that whenever we want to add and populate it with data, we can just run the seed command and it's going to generate and populate our table. And now if we want to test uh creating and adding this data into our table, we can just run npx prisma db and seed.

(24:33) And now we will have that command actually added. You'll see it will say running seed command. The seed command has been executed which means that if we have successfully uh created our seating function correctly when we refresh this we should see the data in our table and that's exactly what we do.

(24:54) So without running any SQL commands without going to the SQL editor over here and running stuff we were able to add data to our table with simple JavaScript. But let's see what would happen if we broke one of the rules that we implemented with the schema Prisma. So for example, uh if I wanted to add two more users, right? I wanted to add Pedro and I wanted to add uh Caroline. Now I'll put here Pedro at example.

(25:23) And for some reason Caroline also wants to put the email uh Pedroacample.com. When I run this command over here, let's see what would happen. I'm going to put npx prisma. And yeah, it broke. Why did it break? Uh, technically it didn't break. It's just that it failed to execute what we wanted it to. And we're not handling errors here. We could and then it wouldn't actually break.

(25:47) It would just deal with the error. But it broke one of the rules that we set on our Prisma schema, which is that the email field needs to be unique. So that's one of the cool things. You can define stuff like that very easily by just putting here the unique decorator and it will mark that field as unique.

(26:06) And I wanted to show you guys an example of where that would fail. So let's just actually uh add more people into our table. I'm going to fix that and let's run it again. And you'll see that now it's very easy for us to populate our table. We just run that command and we can see the data here. Perfect.

(26:25) Now, how would we make a simple API that deals with our Prisma tables? So, I'm going to actually come here to our index.ts that comes with the starter project and we have this server that is running, right? And on the server, we have this app that we can use to generate an API. So, we'll start with the uh most basic API endpoint that we might want to implement.

(26:50) So, I'm not expecting that you know express even though I do think if you're learning Prisma and TypeScript and JavaScript you should know uh Express at least have an idea of of how it looks like and how it works. But the basics of this is that we are making endpoints, right? And for the server that we're running on the local host 4000, we want to create an endpoint that is going to get all of the users in our table. And how are we going to use Prisma to do that? Well, I'm going to make here an a get endpoint.

(27:17) I'm going to set the route to be slash users. I'm going to make an async function that is going to run over here. We don't need the the request, we just need the response. And inside of here to get the users, I'm going to be using the Prisma client that we created. So, I'm going to come over here. I'm going to say import Prisma client from Prisma client.

(27:44) I'm going to create an instance of Prisma just like we did on the seating file. So I'm going to say const Prisma is equal to new Prisma client. And in here I want to create a variable called users. And through the Prisma client I want to await the Prisma user. and instead of creating items, I want to find items on my table.

(28:10) Now, Prisma provides us with five different find functions and they're all unique in their own way. So, find first is pretty interesting. If we just use this and I return here, for example, the list of users as a JSON, we can see that if I actually went here and I went to the localhost 4,000. Oh, I just realized it's my server stopped running.

(28:35) Let me run it again. Uh, so if we went to the localhost 4,000, then we went to slash users, we just made an API endpoint to this / users uh uh route and it returned one entry from our table. But as the the name might imply, the find first function just returns the first item in the table, which is Alice. So that's what that function does. Now let's go into the other one.

(29:06) The second one is find first or throw. And you'll notice that in Prisma this is a common pattern. The idea here is that there's two different scenarios, right? There's the scenario in which you might just want to literally look for some data like the first item like we did before.

(29:24) And if there was no first item then and we were using the find first, if the table was empty and this returned nothing, it wouldn't error. it would actually just return null. So it wouldn't see it as an actual error because it's not expecting anything to be in the table if there's it's okay if no results exist. But on Prisma, you can specify that you actually want to throw an error if nothing is found.

(29:49) So let's see this as an example. This works on every other find function. Um to test this, I'm going to go over here and I'm going to delete everything in our table. So let's delete everything. And because I'm using the find first function over here, if I refresh this now, it should return no.

(30:07) But what happens if I say find first or throw, you'll see that now if I refresh this, all of a sudden it doesn't just return no. It breaks my app. Obviously, we're not handling it here. But if we were, we'll probably put a try catch and then actually return a a good like frontend facing error message to the user. So this is the difference in all all of the versions of Arthrow.

(30:28) And I think that's important for you to know. Now, let's go back to find and let's actually populate our table again. So, what I'm going to do is I'm going to go back to here and I'm going to just seed our table a little bit more and we should see Alice.

(30:47) Actually, we should see Pedro and Caroline appearing here, I believe. Yeah, perfect. Now, we can get the users back. But what other find functions we can use? Well, the user's route is important because it actually can return the entire table. So when we put here find many and we don't specify anything, we just keep the arguments of this function empty. It will return the entire table.

(31:11) As you can see, we only have two users, but it will return the entire table. Now, if I want to find an entry for a specific reason, for example, I wanted to find one user, I can use the find unique function. And this find unique function allows us to specify a condition inside of this uh parenthesis over here that will return back um a user that is unique in the table.

(31:42) Now what can we pass here? What conditions can we execute? Well, you can compare uh any of the fields that are tagged as unique plus the ID. So for example, I can find a user based on their ID by coming to the find unique over here, opening and closing curly braces, putting a wear condition, and then saying that I want to find a unique user where the ID is equal to I don't know six, right? I know that that's the the first user Pedro.

(32:05) So if I do this now, when I return this, we get Pedro. If I change this to seven, we get back the user with ID seven. Right? Now I could also come here and find a user based on their email because email is a unique field. So I could say here find where email is equal to pedro atacample.com and we will refresh this and it will find Pedro. However, I cannot do this.

(32:37) I cannot say name and then look for Pedro. You see it doesn't even allow me to because Pedro is not a unique field. So that's the specific thing with find unique. It is great for finding users based on their area ID or any other unique fields. Now this where conditioning over here is super crucial for Prisma because it allows you to write complex queries using JavaScript.

(32:58) So for example, let's actually make some changes to our table. I'm going to come here to the Prisma uh schema and let's add a couple other fields. Let's put age. age should be an int and no other things we need to like there's no other thing we need to specify with it right now I also want to add an is married field which just says true or false for either the user is married or not and I'll put it as a boolean now I'll also make a nationality field which will be a string now we added those three because it will just make it such that we have more room for us to be able to write

(33:37) complex queries Now if I want to make those changes go through, I need to first migrate. So let's remember what is the migrate command. Well, we need to run npx prisma. We put migrate. We specify that this for is for a dev environment and we give it a name. Now I'm just going to say uh edit name, edit new fields or something like that. And I'm going to run.

(34:06) And we should see that uh when I go to when this is done executed and we go to the public database, we should see the new fields. But here's the thing. We can't actually execute this because we made all of them into required fields. Now, how do I know these are all required fields? Well, whenever you define a field in your schema and you don't put a question mark after the type definition, it is a required field which we just defined age and is married as and nationality as required fields which means that the

(34:42) users that are already there could not be part of the table. So, we need to either delete all the users from our table or make these fields optional like this. But I don't want to make them optional. I I actually have that seeding function. So it's pretty simple for me to just add them again. So I'm going to delete the record over here. We are going to make the migration.

(35:04) I do need to update the seed function to include age and so on. But uh that's that's beyond the fact. We're going to add here uh the migration and I'll be back when it's done. So now it's done. Uh after every migration I need to generate the types right for the Prisma client. So I'm going to run npx prisma generate.

(35:31) And we should see that not only now our table looks a bit different because it includes all the the new fields, right? It includes the age that is married and the nationality, but we should be able to access those in our seed function. So I'll come back here. You see it doesn't even uh it gives us red squiggly lines to notify us that we're missing a couple fields.

(35:49) So I'm going to add a bunch of actual data here. I'm going to add more than just two and I'll be back in a second. So I generated here some more fake data. I'm just going to copy and paste it over here. And um you see each of them it's 10 users. Each of them has actual data. I just use jivvity to generate this.

(36:06) But we're going to run the command so that we can actually see the changes in our table. So I'm going to run npx prism db seed. And we should see that if it's done, the C command has been executed and we should see 10 users here. Perfect. Now, why did I add 10 users? Well, now we can learn how to apply some conditions that are a little bit more complex when we are requesting data.

(36:39) So, like we've learned, we can run the find menu command. And this will look for multiple items inside of our schema, inside of our table that match a certain condition. So if I were to run this then we would only get one user which is uh Pedro right because that's the only user with the name Pedro. But with this wear over here especially when it's not a unique find that we are running we can actually add a lot of things. So let's start with an an easier condition that can return multiple users. So let's imagine this.

(37:09) I want to return only users that are married right that will return multiple users because a lot of them are actually married. So to do that it's easy. I just come here. I change the the field to is married and I put that I only want to find many users which is married is equal to true.

(37:28) If I do that you see now I get every single user that matches that condition. If I were to switch that to false I would get the exact same o the exact opposite of that. So now what if we merge two conditions at once? What if instead of just getting users that are married, I also want to get users that are married.

(37:50) For example, James over here, but the age is greater than 35 because then we get James, but we don't get um Noah, right? Because uh actually let's let's make 30 the age. So greater than 30, Noah is 30, but that's not greater than 30. So we won't get Noah, but we will get James. So to do that, I first put the condition of is married equal to true.

(38:12) And then I can put the condition that age is. And then because I want to apply a comparison to this specific field, I'm not going to put a direct value here because if I did this, then it will just return users whose age is 30, which in this case is just Noah, right? One user.

(38:34) We actually want to apply an additional condition to this specific field. And to do that, you actually apply an object to it. And in here you have a bunch of comparisons that you can apply. For example, you have less than, that's what LT means. Less than or equal to, that's what LTE means. And also GT, which is greater than, and GTE, which is greater than or equal.

(38:54) In our case here, we want to get users that are married and that are greater than or equal to or greater than 30 that the age is greater than 30. So in this case, like I said, we will get, for example, Liam, we're going to get James, but we won't get Noah. So if that's the case, let's see what happens. And yeah, perfect.

(39:14) We got James, we got Liam, we got Lucas, another person, but we didn't get Noah. So this is an example of a a simple but a little bit more complex uh query that you can make using Prisma. Now I want to do something a little bit different. I want to apply an or condition to this um query. So for example, what if I want to get people that are married or so like everyone who's married or people who are Brazilian? That's a case, right? So I don't want to just get people who are Brazilian and are married. I want to get people that are married or they are Brazilian. To do that, what I can do is instead of just

(39:52) applying the is married like this, I can actually use an or condition. And in here I pass an array. And what this array will do is it will detect every condition that we pass to this array will be part of this or. So we will add the items for each condition to the results that are returned.

(40:20) So for example, if we put or and then we just pass nationality is equal to uh Brazilian then we obviously just get back one person right Pedro who is Brazilian. Now Pedro is not greater than actually Pedro is greater than or equal to 28. Let me get someone who is let's make 29 and above. So now the or condition here is that we get age and the age is someone who is greater than or equal to 30. So Pedro is not including that.

(40:48) But Pedro will still be returned here because the condition is that either the person is a Brazilian which Pedro is or they're above 30 which all of this above or equal to 30 which all of these people are. So this is how you kind of apply union between this condition and this condition. Now you could also uh do this which is apply an end.

(41:16) And when you apply an end, this would not return anything because there's no user who is Brazilian but has an age above 30. But if I change this to 25, obviously it will return Pedro because Pedro's age is 25 in this project. So uh this is the opposite of what of doing an or. It's having an intersection between two different conditions.

(41:33) Now one cool thing you can do as well is applying some negation. For example, I could say that I want people who are not Brazilian. So why would I do that? Well, I'm going to remove this end over here. Uh I just want to apply a condition that a person is not a Brazilian. So to do that, I can just turn this into an object and apply the not property to it.

(42:06) And then what I'm doing here is I'm getting all users where nationality is not this value which I believe will be everyone but Pedro. So you see here it's everyone but Pedro. We have every other nationality but that one. So that's how negation works. To build off of this example, we also can actually detect if a field is part of a set of fields or a set of values that we want to search for.

(42:28) So for example, if I only want to get I want to get all the users that are either German, Portuguese or Irish, right? I could come over here and actually do this. I could do this. Put an or over here, pass an array and then put nationality is equal to Irish and then put a comma and then add again nationality is equal to German.

(42:55) Right? And then again, nationality is equal to Portuguese. If I do this, you might expect that it will return every user that is either one of those nationalities. But this seems kind of redundant, right? We're we're checking for the nationality three times. What you could do instead is you could remove this or you can remove all of this and you can use this really cool uh property that you can apply to fields. I come over here.

(43:24) I say I want nationality to be and then I say that I want to get users whose nationality are in a set of nationalities that I want to search for. And then here I just pass the Irish, the German, the Portuguese. And now what Prisma is going to do is it's going to generate a SQL query that is going to look for every nationality that is part of this array.

(43:49) And you see that I'm going to get the exact same return back or result back because uh it's an way easier way for you to write that query. Uh but it still works. So this is the mini crash course I had for different uh queries that you can make using Prisma.

(44:08) I think that understanding all of the different features and um properties that you can apply to your queries make your life when using Prisma a lot easier. Now I'm going to take this course into a different direction now where we're going to first learn obviously the two other CRUD operations that you can apply to any API which is updating and deleting.

(44:27) We've learned how to create and we've learned how to get data but we'll quickly gloss over um updating and deleting users and then I'm going to teach you guys how to apply Prisma into a realworld nex.js project. So, in order to learn how to update a user first, um it's actually pretty simple. The way you do this is we're going to come over here and we're going to put a put request just to make the same request over here, but uh for when you are um updating a user, right? That's what put usually means.

(44:58) Then we're going to put an async property over here. I'm actually going to copy this whole thing just to make it easier. And we're going to change this to put. And when you want to update a user or update a value in your table using Prisma, you use the update function. So use the update function. And as you can see, you can apply this update uh into three different ways.

(45:22) You can update one single item, you can update many items, and we can also update many items and also return the items that you updated. That's what those three mean. But we're going to use the update function. And just for simplicity reasons, I'm going to change this to updated user. So we get it back.

(45:40) But I'm also going to update one specific user. I want to update Pedro. That's the user I want to update. So the user that I uh created for myself, I want to change their age to 35 or something. So when you are running the update function, you have to specify which user you are updating. So the way that works is you go to your schema, right? And you look at this model.

(46:01) We can only detect a specific user by two different fields, the unique field. So the ID or the email. Because if I try to update a user named Pedro, there might be two users who are named Pedro. And then that won't work. So you have to pass a specific unique uh value in your table um to this where uh property.

(46:24) So we're going to update where and then here I'm going to ask for the email. So, I'm going to put here that where email is equal to and then let me look at what email I put for Pedro um pedro@acample.com. So, we're going to update the user who has an email equal to pedro@acample.com and then what I need to do here is I need to update the data, right? I need to pass what data do I want to update.

(46:54) So I'm going to pass here data and I can specify which parts of the user I want to change. Whichever parts I don't specify will remain the same. So in our case, I just want to increase Pedro's age to 35. So I can pass here age and put it as 35 and that's going to update. If you want to update multiple parts of it, I can actually uh update for example the is married to true and that will update it as well.

(47:20) you don't have to pass every new part of this user because some of the stuff will remain the same. Now uh I need to run this uh endpoint. So to do that I'm going to be using just normal curling. Uh you can use something else like you can use Postman or Insomnia or something like that. I'm just going to curl uh the URL of our API endpoint and make it into a put request.

(47:42) Obviously we can't go here like we've been doing because this generates a get request. We want to make a put request. So we have to actually curl it. And when we do this, you see we get back the same user, but now the age is 35 and is married is equal to true.

(48:00) Now the way we know that this worked is we obviously go to our table and we refresh this and we can see that these values changed. Perfect. Now let's do the exact same thing before deleting a user. I'm going be honest, deleting is always easier than updating because deleting is simple to understand. We want to delete a user where a specific unique ID is being passed.

(48:23) So I don't want to delete I don't want to pass any data because that that makes no sense, right? We just need to pass a wear condition. That's the only thing we need. And this function is a delete function. Again, similar to the update, there's a delete many, but we're going to use delete here.

(48:40) And what I want to do is I want to update I want to delete the user where the email is equal to and then let's delete someone random. I'm going to delete Liam. So, delete where email is equal to Liam. I'm going to change this to deleted user and I'll pass this here. Now, let's open this up and let's run a delete request instead of a put request.

(49:06) So, I'm going to curl for the delete endpoint and we should get back Liam. But the way we check this is Liam shouldn't be here when we refresh this. And yes, it worked. But the cool thing about deleting is that when you pass a delete many, you don't have to pass a specific unique ID for it because now you can actually pass a condition, right? I can delete every user who has uh let me see something that is so every user that is above 30.

(49:39) So in this case it will be uh Pedro, it will be Noah and it will be James and Luca. So it will be a lot of people. So I could do that. The way I do that is because now when I'm deleting many users it there's not a specific unique user I want to delete. So I don't have to pass a specific unique value to it.

(49:55) So I'll just pass here I want to delete the user where age and then obviously I have to pass a property here is greater than 30. And you'll see that when I run this now the actually we should update this to deleted users and this is deleted users. And when I run this we should get back account. Now the reason why we didn't get the whole amount is because when we are deleting this it's better to just return account so we know how many users were deleted and you can use this uh later when you're like if you're if you have like a list in your UI and you're clicking the users delete you can then pop up a notification or a toast saying deleted

(50:29) and then you have the count to say how many users you deleted. So now we can refresh this and lo and behold you see all of those users are deleted and everyone with age greater than 30 is now gone. So this is the basics of implementing a simple CRUD and the basics of the Prisma syntax and how to implement it in a project.

(50:49) Now we're we're going to be actually building a project with Prisma. This is super exciting. If you want to check out the code, it will be in the description together with the code for this uh starter package and the point that we're in right now. So go ahead and click the link in the description.

(51:06) You will see the video info kit and uh it will have everything there. So let's get into building the project. Okay everyone. So now let's start building the project. Like I mentioned in the beginning of the video, we're going to be building a job posting website and it's going to be Nex.js using Nex.js 15. It's going to have authentication and it's also going to be entirely using Prisma.

(51:25) So, in order to build this project, I have a starter uh version of the project which you can get from the description below. So, if you go to the description, you click the link, you'll see the final code for this video, but also find um the starter code for this project. So, go ahead, click that link. It's basically just a normal Nex.

(51:43) js app I created using create next app. The only difference is uh there's going to be some CSS in here that I added to the global.css. Um, also the favicon I made um an icon for the website and I put that same logo into the public so that we can use it on the website as well.

(52:03) So that's all that you have to keep in mind. Also, we have this package.json. And on this package.json, it contains all the dependencies and dev dependencies we're going to be using on this project. I already have them here with their respective versions. uh when you download or when you clone this package, you just go ahead and run npm or yarn, whichever uh package manager you're using, you run npm install and it's going to install all of those and create the node modules folder, which is already over here, but it will add those packages to that folder. So, let me go over real quick what are the

(52:36) dependencies, libraries, and stuff that we're going to be using for this video. So, for the project, we're obviously going to be using a Prisma. So we have here Prisma client. We're also going to be using Prisma extension accelerate which is just going to help us set up Prisma in our Nex.js app.

(52:56) We're also going to be using Next O for authentication. Going to be using version five for that. So if you want to install the package from scratch, you'll have to make sure to specify that you you want to use version five because it's still in beta. But in order to use Prisma and connect it to our O provider, we're going to be installing this O Prisma adapter that connects a Prisma database table with Next OJS as it's now known. Um, so we definitely need that.

(53:24) And also, we're going to be using date FNS just for uh some UI dates that we're going to be creating on the project. And I think that's pretty much it. All the other ones just come with an Nex.js project. So, let's close this off and let's run the project. I'm going to run npm rundev.

(53:43) And we should see that when I go into our uh Google Chrome over here and I run localhost 3000, we should be able to see our project running. Perfect. Now, what I want to do is I want to take a look at our tables here. I already made some small changes to them uh since the last time we talked about uh but I'll just tell you guys what you have to do. just delete all of the tables uh that were there.

(54:08) The user table that we created, just delete that. You delete it by clicking on this button and clicking on drop. And then you just delete it. Also, the Prisma migrations folder. We're going to delete this as well. Uh the reason for that is because we're going to generate it again through this project.

(54:27) We're going to use the same uh neon database we used for the uh the other part of the video for this project. So, I would delete that. Now we have to use Prisma to create our Prisma model and set up Prisma on our project. So we've we've done this before. We already have Prisma installed. So I'm going to run npx Prisma init. And what that will do as you remember is it will create av file which is going to help us input our database URL which again we can get it from the project that we're just coding on. I already have it set uh next to me over here. So I'll just paste it. This is the URL for the table that

(55:02) we have on Neon. But also we want to we we come with a Prisma folder and inside of it there's a Prisma schema. Now on this schema previously we didn't have an output uh but now I want to keep this output. You see it's outputting to the app generated Prisma.

(55:27) This will make it such that uh we're going to have a folder on our app folder over here which is going to be called generated and inside of it it will contain all the generated types created by Prisma client um inside of it. So we're going to keep this. We don't want to deal with this uh like we did on the previous project. Uh we're going to keep it this way and I think the rest should be fine. Provider should be Postgress and so on.

(55:46) Uh now I want to talk to you guys about something. Since we're using Next O for this uh which again it's a it's a authentication provider for our NexJS project um we need to set up some of the models that they give to us uh that are related to a user's authentication state.

(56:11) So they have support for Prisma and they provide to you actual models that you will then have in your database automatically that keep information about a user session, a user's account and so on. So you can go to the the OJS next O website and copy the models that they provide for us that they specify for us. But I'm actually just going to copy and paste it here cuz this is what uh they give to us. I'm I didn't write this. So the first thing they give to us is a model for an account.

(56:40) Uh this is just an account that a user has when they create an account and authenticate on their website. It includes stuff like their ID, uh their type, the provider that they used, which we're going to have to use GitHub for this video. We're going to the user is going to be able to log in with GitHub, and stuff like the refresh token, access token, and so on.

(56:58) So this is what the account type looks like. We're then going to have the session type. This is the session model. It will have a table that will contain all the different users and their sessions, including their session tokens and when they expire. So, this is really cool because you'll be able to see the sessions for each user.

(57:21) Finally, we're going to have a verification token model, which like the name implies, it's for authentication purposes and it will include verification token information. So, we have those three models. We don't have to worry too much about them. Just know that they're necessary for when we create the authentication for our project.

(57:40) And now we can actually start creating the tables and models we're going to be um using in this video. And starting with the most basic one, which if you think about it is similar to what we have used in the previous part of this tutorial. We're going to be using and creating a user. So a user is going to be a table.

(58:02) uh it's going to be a model that it differs a bit from the model of account or session and so on because this over here keeps information about an accounts or a user's off state. A user on the other hand is going to keep track of some other information about the user. for example, their email, their name, if their email is verified, their any profile picture, um, and also it will relate, it will include information about the user's accounts, the user sessions, and so on.

(58:31) So to do that, we're going to come down here and we're going to create a model called user. Now on user, we're going to have an ID. The ID is going to be a string and it's going to be at id. And as you might remember, it will be at default. And instead of doing auto increment like we did before, uh we're actually going to be using a different function. It's going to be called CU ID.

(58:53) And CU ID will just generate a random ID, which is a string of like a bunch of numbers. So this is how you generate that ID. Now, after this, we're going to be wanting a name. So each user will have a name, and it will be optional just by default. Then we're going to have an email, which will also be a string that is optional, but an email has to be unique.

(59:18) Then we're going to have an email verified. And this is optional. You don't have to put this, but this is just the time in which they verify their email. For our project, we're going to be using GitHub. So this isn't really applicable, but I usually when I create a user, I have this in case I make sure that they are allowed to authenticate using email and password.

(59:37) Then we're going to have an image. And then image is just a profile picture. It will be a string because it's going to be a link to their GitHub profile picture. Then we're going to have accounts. Now, this is where we introduce a new topic in Prisma and it's a topic that isn't Prisma specific. It's SQL specific and it's about relations.

(59:59) So, a user has an account by default, right? So when you create a user, you authenticate that user, you authorize the user to get into the website and an account, a session, a verification token is created. So what we need here is we need to establish uh the fact that a user can have multiple accounts. So we're going to set the type of accounts here to be account and array.

(1:00:25) Now, we're also going to do the exact same thing with sessions. So, we're going to say sessions and session array. Now, you see I'm getting red squiggly line here. And it's because when we associate a type a model with another model like we're doing here, we have to establish the relationship by coming to the other two models like the account and the session and establishing that relationship by defining that other model here.

(1:00:54) So for example we associated the account model with the user model. So on the account model I have to define a user which has a user type and I have to establish that relationship between them. So for example what relates a user to the account. So let's think about it. What part of the account model can help us identify when we are looking for a user that owns that account? Well, a user has a user ID and guess what the account also has a user ID.

(1:01:26) So what we can do is we can say that this user relation here so we use the relation um tag and we say it's compared it's associated based on the fields and here we put the field in the account model that we will compare to the user model and in this case is the user ID. So we say the field is this and the reference on the other model is the is the ID.

(1:01:52) So it's this. This should be equal to this. And that's how we establish a relationship. We can also specify on delete. So what is on delete? It means if one of the two an account or a user is deleted, what happens to the corresponding account or user in the other table? Right? If a user is deleted in the user table, what happens to the user with that user ID in the account table? So we put cascade meaning that it will delete on both tables. So this is how we establish that relationship.

(1:02:23) And we're going to do the exact same thing on the session over here. We're going to establish the same relationship. And since we also have a user ID here, it will be the exact same thing. So perfect. Now we have learned first of all associations and relations. And we have established that relation.

(1:02:43) And this is pretty much what a user will look like. Now let's create the auto models that we need on this project. So the main core of the project is that we are a job posting website. So what do we need in a job posting website? We need a job. So we have to create a model that is going to represent what a job looks like.

(1:03:00) So I'm going to create here a model and we're going to call it job. In here we want to have uh the same thing as we did for user an ID. same uh like it acts the same way. Then for a job we need a title which is going to be a string. We also need a company name which is going to be a string.

(1:03:22) We need a location like a city slash country that we need and it will be a string. We need a type. Now a type is going to be like is the job part-time? Is the job full-time? Is the job a contract? So that will also be a string. Then we need a description of the job. So when a employer posts a job, uh they need to write a long description.

(1:03:47) So we're going to specify that this is actually going to be a DB.ext just like this. And then we're going to ask for a salary. A salary is going to be a string, but it's also optional. I think like jobs might sometimes don't put the the salary range, right? Or the salary as a whole. So we're going to make that optional. Then we want to get some stuff that is important for when a job is posted.

(1:04:12) So for example, when is when was this job posted? So posted at now that's a date. So I'm going to put here a datetime type and we're going to default it to now. And what that does is it basically creates the current uh whenever we add a new job, this is going to be automat automatically filled in by the current moment so that we actually have information of when the job was posted. We're also going to have a posted by.

(1:04:37) But then let's think about it. If a user can post a job, right? Then this posted by type field over here is going to have a type of user. which now gives us an error because we don't have a relation between between a user and the job. So let's think about it. Is it that a job has multiple users or is it that a user can have multiple jobs? Is the association such that user is the one model that can relate to many jobs or is it that one job can relate to many users? Well, if you think about it, yes, a job can relate to many users, but when we create our website, we want to keep

(1:05:23) track of all the jobs that a user have applied to. So, the association is going to be such that when a user ha creates an account, we're going to keep track of all the jobs that they have applied to. So, we're going to keep track of all the jobs.

(1:05:43) So, I'm going to create here jobs and I'm going to create a job type that it's an array. Now to do that I'm going to establish that relationship over here. So I'm going to say at relation and we're going to give a name to this relation which is a bit different from what we did over here. So the name of this relation is going to be posted by or posted jobs and uh I just want to know who posted this job which user posted it.

(1:06:08) So we're going to set the fields to be posted by ID and we don't have that field yet. So we have to create it over here. That will be a string because when we post a job, we're going to pass the the user ID. When we create a job, we'll pass the user ID. But we don't have to pass all the users information.

(1:06:30) It's going to be autogenerated based on that ID. So then we have to pass the references. and the references will be the ID type on the user model. This means that now we have successfully established a relationship. But wait a minute, it's still giving us red squiggly line even though we did exactly what we did with this uh with the session and account on the user.

(1:06:52) Well, the session and the account relations don't have a specific name to it. Remember, we didn't put this this name over here. So when you put a name to it, you have to also establish the relationship or specify the relationship on the other side. Perfect. Now we have successfully established the model for a job. Now the last model we want to create is an application.

(1:07:16) So I'm going to say model application. And for the application again we're going to have an ID which is going to be a string. We're then going to have a job ID which is also going to be a string. We're going to have a user ID meaning which user applied for this job which is going to be a string. And we're going to have a status.

(1:07:42) Now what is a status? A status is going to be uh basically what is the status of this application. Is it appending? Is it accepted? Is it rejected? Is it a user reviewing? But this website won't be extremely complex in the sense that it's an like the user will the employer will actually go ahead and establish a contract and all that kind of stuff.

(1:08:03) We're going to keep it simp because the whole point of this video is not to teach you guys how to build this project is to teach you guys Prisma. So, we're going to keep it as simple as possible. We're going to default it to pending. So, we're going to default it to pending. And we're going to keep it like that. Uh all applications that are that are published are going to be pending.

(1:08:21) But if you want to expand on this project, you could then add the ability to change it to reviewing um accepted uh and rejected or or so on. So you can do that as an extension to this project. Then we want to have an applied at and this is going to be a date time and we're going to default it to now similar to what we did above.

(1:08:53) So default now, now let's think about what um an application needs. So an application connects two different models. Uh it can it needs a model of a user and it needs a model of a job because it's the bridge in between a user getting a job is the user making that application. So in an application, we need to keep track of information about a specific job and a specific user.

(1:09:17) So we'll have to establish relations for both of those models. Let's start with user. So I'm going to say user over here of type user and I'm going to say it has a relation in which the fields user ID references the field ID. Right? Because we created the user ID field over here. We'll do the exact same thing for the job and the job ID.

(1:09:43) So we'll say job and job and we'll change this to job ID and the reference here should still be ID. Now it's still red squiggly lines again because we have to establish them on both the user and the job type. So over here for the job type I'm going to say applications and then application array. And for the user I'm going to say applications and then application array.

(1:10:09) Perfect. Now one thing that I want to do is I want to enforce the fact that a user can only have an application to a job once. Meaning a user can't apply to the same job multiple times. That's an important factor to consider. And that relates perfectly to an interesting thing you can do in Prisma, which you've already seen up here, but I didn't explain because I was going to get to it.

(1:10:41) You can establish a unique enforcement to a relation. So, for example, in here, he's saying that a verification token can only exist if the combination of the identifier and the token are unique. This is known as a composite unique constraint on this model. So in this case over here we have to implement a composite unique constraint on the application model because it will ensure that each combination of a job ID and a user ID is unique in the database.

(1:11:11) And to do that you come down here you apply a double at unique and you pass an array of the two reference values. So we want to make a unique combination of a job ID and a user ID so that a user can't apply to the same job twice. And um finally uh we're pretty much done with all of our Prisma code for this project.

(1:11:37) We have created three different models ourself and used the models that uh ojs or next o whatever you want to call it um are going to use to let our users authenticate. Now what we have to do is we have to migrate this db into our tables. So I'm going to say here p npx prisma migrate dev and then d-name and I'm going to give it a name.

(1:12:08) It's going to be in it so that we initiate this project. Going to press enter. It's going to load the env variable from ourv file. And if everything goes right, it's a big uh creation. We're going to be able to see those tables in our schema. So let's go back over here and let's refresh this. And all of a sudden, boom, you see all of the um tables.

(1:12:33) Now, one thing I need to let you guys know in case you run into this, if you didn't delete the Prisma migrations folder uh table before and you still had leftover migrations from uh the previous project that we've built, you might want to run this command over here, which is going to allow you to reset the migration table.

(1:12:58) So if you if you run into issues where it's not allowing you to cuz you have migrations already, you run that and it's going to work. So we have successfully created here our tables. So let's start building the code for this project so that we can use this models. And before we do that, let me just actually generate the types. Right? So I'm going to run npx prisma generate.

(1:13:20) It's going to generate first of all a generated folder on our app because that's how we established over here where the types are going to be at. And when we create our Prisma client, it's going to use those types. Perfect. So, I'm going to close this and let's get started. So, let's start with the um navbar. So, over here on our layout.tsx, I want to make a navbar.

(1:13:45) And the reason why I want to make it first is just because um we're going to use that to be able to allow a user to sign in into their accounts. So in our layout.tsx, I want to wrap this uh the children around with a main tag. And I'm going to put an entire project inside of this main. Uh we're going to have here the navbar component that I'm gonna create. I haven't created it yet. And I'm also going to apply some minor styling to this main tag.

(1:14:13) So we're going to make it into a container. We're going to set MX to be auto, px to be equal to 4, and py to be equal to 8. Now, I'm not going to be writing styles in this tutorial in the sense that I have the styles done for the project.

(1:14:31) But this isn't at all the purpose of the video and I think it will make the video very bloated if I explain every class name and every style that I that I make. I have tutorials on tails tailwind CSS if you want to learn it. But I will copy and paste the styles that I've already written and show you guys doing that. So if you want to get the styles, go ahead and click the link in the description or just copy from the screen whenever I make them. So perfect. We have here our class.

(1:14:52) I'm also going to add here a div that is going to wrap around our navbar and our main. And on this div, I'm going to add a very basic class name which is going to set a minimum height of the page to be the screen size. And it's also going to set the color of our background of our website to be gray 50. So now we have to create this navbar component.

(1:15:19) So we're going to go back here to our project and I want to make a components folder. So I'm going to make that components folder here outside of the app. We're going to make components. And in here I want to make the navbar.tsx component. Now the navbar, it is going to be very simple because there's only going to be like three links or so. We're going to make here a function called navbar.

(1:15:44) And inside of it, we want to return some UI. So I'm going to return here a nav tag. Inside of it, I'm going to add another div and another div. So two divs, one inside of each other. Then in this second div, we're going to put the first part of the website. And the first part of the website it is going to be if you remember in the intro there's like a logo uh with the name of the website over here and then there's the links and the sign-in button over here.

(1:16:14) So what we're doing here first is the left side of the page. So I'm going to add another div and on here we're going to have a link which is next.js component and we're going to put an image with the logo. So let's put the image over here. It's actually a component. So image I need to import that. So import image from next image. And the image component is going to have a source and the source is going to be the logo.png file that I've already added to the starter of this project.

(1:16:47) We're also going to apply an alt to this image because you always should for for accessibility reasons. I'm going to say jarboard logo. I'm going to set just a minor width of 40, a height of 40, and that should be good to go. Now, on this link, we want to put a path to where you go when you click the logo.

(1:17:12) I'm going to take the user to the homepage whenever they click the logo. And I also want to put uh a span right next to the image, which is just going to write the name of the website, which is very generic. I'm just going to call it job board. Now I'm giving an error over here just because I haven't imported the navbar component.

(1:17:30) So we'll import it and we should see that appearing on the top left. Now on our navbar we have finished the left part of the navbar. Let's start building the right part of the web navbar. So on this div over here, we want to escape out of it and we're going to make another div.

(1:17:50) This other div is going to contain a link to the browse jobs page. So, the jobs page, we're going to make an href, which is going to take us to the /job page, which we haven't made it yet, but we're going to make it. And also, we want to make a a page for a user to be able to post a job and also see their application. So, a dashboard.

(1:18:18) So, we're going to have here a post, a job link, and a dashboard link. So the links for that is going to be jobs slashpost and this one is going to be just slashdashboard. Now we should see them over here. The obviously the UI is very unpredictable but we're going to add the styling soon. Um so don't worry about it. But we have now those button those links.

(1:18:44) We also want to add a signin button. So we're going to create here what is going to be actually a link. It's not going to be a button. We're going to say sign in and we're going to put an href to the following link slash off slash sign in and it should appear over here as well. Now I want to copy and paste all of the class names for the styling of this navbar so that we can actually see a cool looking navar appearing. Perfect.

(1:19:12) I just added here the uh styling. You can copy all the class names. I'm going to just roll over a little bit slowly so you guys can see. But at the end of the day, you can just get all of this code in the description. And this is how our navbar looks now. It looks a lot better.

(1:19:30) You can navigate between the different routes. Obviously, a lot of those routes don't have any pages in them, but we're going to create them. So now, let's close this off and close the navbar. We want to start building the sign-in page for our website. So that page is going to exist inside of our app. It's going to be a route and we're going to create it. We're going to call it O over here.

(1:19:52) So, we're going to create an O folder and the O folder is going to have a signin folder because if you remember uh not singing or whatever that is, it's sign in. If you remember on the navbar, we navigate to /signin. So, that's how we make that route. Then, I want to make a page.tsx inside of here.

(1:20:20) And then in here we want to export default function and call it signin page. Now on this page we want to return a div and inside of it we'll put another div and again another div. Now on this third div over here we're going to have the top part of this off page. We're going to have a little H2 tag just saying welcome to the job board or whatever I call this website.

(1:20:50) And then below this going to have a little P tag that will let us know that you need to sign in to post or apply for opportunities. So we'll say sign in to post jobs or apply for opportunities. And then below this, we can get out of this div. We want to create another div with the actual uh signin form. And in this case, our sign-in form is just going to be a button to sign in with GitHub. Of course, you can add other providers if you want to.

(1:21:23) You can check out the OJS um documentation for adding Google. If you want to, it's extremely similar as what we will be doing for GitHub, but um I just want to keep it simple. So, we're going to have GitHub for this project. So to do that, I'm going to have a button which is going to be the sign in with GitHub button.

(1:21:40) So in this button, we want to first have an SVG with the GitHub logo. So to do that, I have here the SVG generated. This is how it's going to look like. Let me go to the sign-in page so you guys see it. Uh yeah, it's hard to see, but you can kind of see it over here, right? So yeah, that's the SVG for the GitHub logo.

(1:22:02) And then I want to add a sign in a span that is going to say continue with GitHub. Perfect. Then just to make it a little bit cool and more authentic, I want to add this little div over here at the bottom which uh websites usually have. It's saying something like by signing in you agree to our and then I'm going to put a link to our terms of service.

(1:22:29) But we don't actually have any terms of service. So I'm just going to make an a tag instead of a link. And we're going to say terms of service. And I'm actually going to put this below here. And then I'm going to put end. And then I'll put another link to our privacy policy or whatever sounds uh fancy. So terms of service or privacy policy and it appears like this. Perfect.

(1:22:54) Now let me apply the styling cuz you guys barely can see anything. So I'm going to be back in a second when I added all of the class names. And just like that, I added all of the class names needed for this page. You can see it's I'll scroll down a little bit slowly. I can also get over here so you guys can see everything. Really guys, just check the CSS in the description. You don't need to copy everything manually.

(1:23:18) Um, but now this is what it built. It built this little simple card in the middle of the screen and it has the GitHub button and it has those other privacy and terms of service policies as well. Now, what happens when I click on this button? Absolutely nothing. And that's exactly what we want to change.

(1:23:37) we want to start implementing the ability for a user to be able to sign in and authenticate with GitHub. And to do that, we're going to be having to create some stuff related to Next O. So, like I said, we have installed over here uh a couple packages related to Next O. Uh first, we installed the Next O package and remember it's in version five, which is in beta and we also installed O/Prisma adapter.

(1:24:00) That's going to be crucial for us. So when we are going to set up authentication using ojs, we need to first create an o.ts file at the root of our project, not inside of the app folder. Inside of this file, we're going to set up the entire infrastructure for our authentication. So I'm going to say here import next o from next o.

(1:24:23) Then I'm going to say import github. And this is actually something that comes from next o. So we'll say next o slashproviders slashgithub and then down here we want to import the prisma adapter from o/prisma adapter. Now to make this work we need to have access to the prisma client and to make prisma work in nex.

(1:24:53) js especially with typescript there is a thing that you always have to do. We're going to create a lib folder in our root of our project. So I'm going to create here lib and inside of here I'm going to make a prisma.ts file. Now in this file I'm going to paste this code. Now this code over here is the code given to us by Prisma to set up Prisma in uh next.

(1:25:18) js project and it basically sets up the type definitions for Prisma using the extension accelerate package that I mentioned in the beginning of the video. Um, whenever you want to set up a Prisma client or Prisma object in your project in Nex.js, always just copy and paste this and you don't have to worry about it. It just sets up the types for you so that you can use your Prisma client.

(1:25:37) Now, we can go back to our off.ts and we can import Prisma from at /lib/prisma and we have access to the Prisma client. Now down here we want to export const and then call the next O function. Then what we're going to get back from this is we're going to get back a couple things. The first thing we're going to get back is an O object.

(1:26:04) It's a function actually which we can use to call whenever we need to know if a user is authenticated or not. So we literally just call this O function and it gives us back the info if a user is signed in or not. So it's very simple for us to get the session.

(1:26:20) Then we also need to have access to the handlers and you guys will understand why later. Then we need a sign in and a sign out function. This is all provided to us by next off and obviously as you can see we're going to be using this to sign in a user or sign out. Now inside of here we have to specify the session strategy and we're going to be using JWT for it.

(1:26:41) So I'm going to say strategy is equal to JWT. Then below here we want to specify what providers are we using to authenticate. So providers obviously by the name it implies that we're using GitHub, right? And if you're using G GitHub or Google or Outlook or whatever you're using, you'll pass all of them into the list of providers.

(1:26:59) In our case, it will just be GitHub. Then we want to pass on an adapter. Now, why is an adapter necessary? Well, it's only necessary if you're using some sort of ORM that um will store information in a specific database. So in this case, our adapter is Prisma because that's the ORM we're using in this video.

(1:27:18) So we want to pass the Prisma adapter and and in the same time, we want to pass the Prisma client inside of here and it will automatically handle all of the authentication, creation, all of that stuff for us. When a user signs in, then we want to implement a callback. So what is a callback? Well, callback functions are functions that you can define to control what happens during the different stages of authentication. In this example over here, we're going to set up two different callbacks.

(1:27:50) The first one is going to be an a JWT callback. And I'll just make it like this. And then the second one is going to be a session callback. Now, what do they mean? Well, this JWT call back is a function that is going to run after a user signs in and only once.

(1:28:13) So when we run this, I want to first of all just make it async. So I'll make it async like this. I'll have access to the information about the user's JWT. The name JWT is not uh invented. It's a name of a function that we can set up as a callback function from uh next off. So we have access to the JWT token and also the user info.

(1:28:35) So when we set this up, we want to check only if the user actually is available at login. We want to add some extra properties to the JWT token just because we want to make sure that we have access to those properties when the JWT token is stored in the browser. So, I want to add a couple things to this token.

(1:29:01) If you're not familiar with JWT authentication, then I wouldn't worry too much about this. I would maybe just watch another video on JWT. I have videos on it. It's like a whole different topic. But to be honest, in today's day and age with something like next off, you don't really have to worry that much about it. Uh, it does everything simply for you, but we're just storing some information into a token.

(1:29:18) And in this case, we want to keep track of the user's ID. So we'll create a type called ID on the token object and we'll set it equal to user do ID and we'll do the same thing for name so that we have access to that as well. So we'll say user name. Perfect. And in the end we just want to return the token.

(1:29:40) Now for the session we'll be able to access the same kind of information over here. So the first thing we can access is the session and also the token. And because we now have uh the ID and the name in the token, we can assign those values to the session as well, which aren't things that are by default in the session object.

(1:30:01) So what we want to do is if uh the session do user is valid, then we want to set session do user and then you see that the only things we have access to over here is this. We want to set the ID equal to the token do ID and I want to set the I want to cast that as a string and we want to do the exact same thing to name.

(1:30:32) So we're going to say equal to token dot name and also a string. In the end we're just going to return the session and this is going to be fine. Now, whenever you set up a next o uh object like this, uh I highly recommend always setting up the JWT and session callbacks just like we did. In summary, we're just enriching the token on the GWT call back over here so that it has more user info so that we can then use it later in our app.

(1:31:03) And in the session, you're we're just ensuring that the session returned to the client has the same ID and the same name from the JWT token. So now we're pretty much done with this setup page. We're going to close this off. And I want to also just set up something real quick. We want to create a middleware file. Now based on what uh Next O recommends, we just have to export here the O file from at / the file that we just created.

(1:31:33) And we're going to change its name to middleware so that Nex.js recognizes it. So, we're going to say middleware and they're going to handle it all for us. So, now let's go back over here. We need to do one final thing. When we sign in with GitHub, Next O is going to reload us or redirect us back to somewhere in our page. And to have that working, we have to set up the API route.

(1:32:00) And on this API route, we have to have a folder called O. And on this o folder we have to have a folder called triple dots next o. And on this folder we have to have a file called route.ts. This is just based on pure like pure what next o requires us to do. And inside of this route.ts we're going to use the handlers function that we got on this offs file.

(1:32:28) This handlers function over here. and we're going to use it by importing it. So handlers from at o and all we're going to do is this is going to give us access to a get and a post request that we're going to create on this file. So we're going to say handlers and we're going to get back over here the get and the post. Perfect.

(1:32:51) Now I'm going to close this off and now we can finally start implementing our signin functionality. To do that, I'm actually going to create a new file inside of our lib over here, and it's going to be called O. This is going to include all of our server actions that are going to actually allow user to sign into an account.

(1:33:10) So, if you remember, we had here the sign in and sign out functions from at /. So, we're going to use those. We're going to get a sign in and sign out. Now, in here, we're going to export const two functions. The first is login which is same thing as sign in but I'm going to keep it like this so we don't have double imports and the second one is going to be a log out. So we'll just call it logout.

(1:33:40) Now on this login we're going to await and we're going to call this sign-in function. Now the sign-in function we have to pass first what is the provider we're using and you see there's a billion of providers that Nex off allows you to use but in our case we're going to be using GitHub.

(1:33:58) Then we need to pass where is our user going to be redirected to after the user signs in with their GitHub provider. So we have to put a redirect to uh route over here and we're going to put just redirect them to the uh empty like the the index of our page. So we'll keep that. Then we're going to come down here. We're going to await and we're actually going to sign out.

(1:34:18) Now, now sign out doesn't really have to pass a provider because you don't sign out with GitHub. You just sign out. But we do want to put a redirect to so that the user knows where to go. And the redirect to that I'm going to put is the O slashs signin because when they sign out, I want them to sign back in.

(1:34:41) Now, we can come to the signin page and we're going to add an on click to this button. just going to say on click and we're just going to call the sign in function. So I'm going to say sign in and we need to import this function. So I'm going to autoimp import it and then we want to log in. So I'm going to call the login function and this should be good to go. Now it won't work yet.

(1:35:06) And the reason for that is because there are a couple environment variables that we have to set up. The first one is this uh o secret that is required by uh o.js and the way we generate it is we just run npm exact o secret. We press enter and it will automatically create a env.lo and add this uh o secret for us.

(1:35:33) We don't have to touch it but uh it generates it for us. Then what we need is we need to add a github o provider. So to do that, you have to go into your GitHub account. You have to click here and go to settings. And then in settings, you go to the developer settings.

(1:35:54) Now, I'm not going to click on it uh and show you guys because um when I click on it, actually I will, but I won't click on oath apps. You just have to go to OOTH apps, and I just have a bunch of them, and I don't want to show all of my OOTH apps, but you click on this, and you click on new OOTH app. Then when you do that, you'll be granted with this page over here.

(1:36:12) Uh, and you want to generate a client ID and an client's secret. I already have them. They're over here. Uh, you see, I edited this 3 hours ago. Um, and I'm going to be using them. I'll delete this project right after so you guys don't use the same as mine. But when you're done getting your client ID and your O secret, we go back into our um.env.local over here and we add both of those.

(1:36:36) So I added here an Ocore Github ID and Ocore GitHub secret and just having them like this. You don't have to do anything else to it. Just name it exactly the same and keep them on the env.lo and Nox next off will automatically get those and handle uh adding those to the GitHub O provider. So now we're pretty much good to go to allow a user to sign in.

(1:37:01) Let's see if that works. So I'm going to come here to our project. We are seeing an error. I'm going to see why it's erroring. So I found the issue. The issue is that on our o.ts, uh we're not going to use the Prisma instance that we created over here. We're going to use that for everywhere else in our project.

(1:37:20) But in here, we actually want to manually import the Prisma client from the app generated/prisma. And then we want to create an instance of it. So Prisma is equal to new Prisma client. And now if we go back here, you should see that we see again the sign-in page. And that error is gone. So now we're able to sign in.

(1:37:48) Let's check that out. I'm going to click on this button. And I'm already signed into GitHub. So it should be automatic. And all of a sudden I'm in the homepage. Now we don't see anything visual yet. But the way we can check to see if the user actually signed up is we can go and see if any data was added to any of these tables.

(1:38:07) For example, as you can see, a user was created for Pedro with their email, their profile picture from GitHub, and other stuff as well. Not only that, but an account has been created as well. So, we know that the user has successfully been created. Uh, now we just have to have some UI to show that this user is actually authenticated or not. So let's go back over here and let's make that happen.

(1:38:31) Starting off with the uh the navbar which is something that we have already made. First of all like I don't want to see the sign-in button when the user is already signed in. I want to see this become a sign out button. So to do that I'm going to want to have access to whether or not the user is signed in or not. So I want to know about the session of the user.

(1:38:52) But this navbar over here is actually a client component. I know we haven't made it into a client component yet. We're going to do it right now. But it is a client component because if we want to have a button to log out the user, we're going to have to have an onclick on that button. So this component has to be a client.

(1:39:10) Now in client components, it's not easy for us to just get the session of the user. To do that, we actually have to create a session provider that we're going to distribute so that we can access the session in every single client component. To do that, we're going to make here a file called a session provider. It's going to be a tsx file. And this is actually a provider that is given to us by next o.

(1:39:36) So we just copy and paste it over here. It just creates a component called session provider which takes in a session and then provides it to every other children inside of it. We want to use this in our layout component over here because in the layout we can wrap around our entire project with this session provider and now all of our client components or every single component will have access to it as well.

(1:40:04) But the thing is in order to pass a session provider we need to provide it with a session right and the way we're going to do that is by using the actual way to get a current session which is our o function. So in the root layout over here this will become a server component. So it's going to become an async function. We're going to get the session by using the o function.

(1:40:30) So I'm going to create a session variable set it equal to await o and remember o over here I'm getting it from the o file that we created and now I can pass this session to the session provider and if I want to access uh the session information of whether or not a user is a or not in a client component all I have to do now is I can use this use session hook from next o and it will automatically provide it with the necessary data. Now I'm going to change this value to be called session that we can use it uh more properly. And the way

(1:41:04) I want to differentiate this is let's think of the links and buttons that um should appear when a user is authenticated and the ones that shouldn't. So for example, browse jobs. Sure, a user can browse jobs even if they don't have an account or they're not logged into their account. But to see a link to post a job or to the dashboard or to sign in doesn't make sense if the user is not authenticated.

(1:41:27) Uh so what I want to do is I want to wrap around uh this two over here. Uh forget the ne the sign in that that's actually what you want to see when you're unauthored. And in here I want to check is session valid. Is it null? Because if it's not null then we want to see those links.

(1:41:51) And since they don't have their siblings I need to put here a fragment to wrap around them. Now I need to put an else condition. And the else condition is that the session is not active, meaning they are unauthenticated. Then I can put this link over here. So we see the sign-in page. And you can see now that if I go over here, we don't see anything because it's failing something. Oh, I just need to refresh.

(1:42:17) But yeah, we see all of this, but we don't see the sign in button. So what we want to do now is allow the user to sign out. So So below this dashboard over here, I want to add this button. It's going to be a button that has the following class names if you want to apply the styling and it says sign out and again it's inside of the session gating.

(1:42:34) Now the logout button over here we want to import that function. I don't need to make it into a callback function. I can just do it like this and it should be fine. And now when we go over here we should see the sign out button instead of the signin button. When I click on that, we should now be redirected to the O/signin.

(1:42:58) And if I refresh this, we should now see that those states updating. Now, let's create the posts uh the the job post um component. So, we have a page that is going to allow us to browse through the jobs, but we also to actually see any data, we're going to create the create job route first. So, in order to do that, I'm going to create here a folder in the app folder. It's going to be called jobs.

(1:43:25) And in this jobs folder, I want to create uh not a file, a folder called post. And this folder is going to have a page.csx. And then when we create a post, it's going to be inside of this page. So, we're going to create here a default function called post job page. And in here we want to add the following UI. So you see here we have the UI. It includes a title for this page that says post a job. It includes a form.

(1:43:59) And the the biggest part of this page is going to be a form. The form includes an input for the job title. An input for the company info. Uh an input for the location which the job is. an input for the job type which turns into a select with um four different option or five different options, right? And then also a text area for a description um an input for the salary which is optional and a button to submit the form.

(1:44:25) Now the reason why I just skipped me making that is just because uh I don't want to waste time with making the UI. So we're going to sign in with GitHub over here. We're going to go to post a job and we should see that this is how the UI looks like. So looking great.

(1:44:47) Now I want to show you guys how to implement the functionality to actually create a job in our table um by using this form. So to do that we're going to come back here and this is a form. So we need to create a handle submit function for it. So at the top over here I want to make a const uh handle submit is is equal to async and I'm making this function that is going to take in an event which is a type of form event HTML form element and inside of here I want to first prevent default so the page doesn't refresh and Then we want to import this from React and we want to get back the form data from this form. So to do that

(1:45:32) I'm going to say form data is equal to new form data. Then I'm going to say E.curren target and through that I can construct this object called data. Now this object is going to allow us to uh define how the data is going to look like when we send it to our uh Prisma client.

(1:46:09) So if we look at the model we created actually we can look at over here right a job has let me go down it has a title a company a location a type a description and a salary the posted at posted by and posted by ID and the applications it's all things that we don't necessarily need to pass ourselves so when we create a job we're going to do it like this we're going to say the data is going to be a title which is going to we're going to get it through the form form data object by saying get and then we specify that we want to get the title uh input with the name title or the ID title. So we say title over

(1:46:40) here and we can do the exact same thing for the other ones. I can do it for example I'll do it here for the company and we can get the company. Now when you're done adding the other ones uh just the exact same way we can send this data through an API request. So, we're actually going to be using API routes for this.

(1:47:07) We're going to make here a new API route in our API folder is going to be called job or more specifically, we can just call it jobs actually. And in here, I'm going to make a route.ts. Now, on this file, I want to export an async function called post because this is going to be a post request since we are going to be sending some data to our table. Then we're going to get a request and inside of here the first thing we want to do is find out and validate if a user is authenticated or not. This will ensure that not only we are validating that on the on the client but also in our server endpoints.

(1:47:41) So we're going to do that by again using the uh O function from next O. So, we're going to import that and we should be good to go because now if there isn't a session do user uh not session storage a session do user and or or in this case let's actually also ensure that there is a session do user do ID then we return a new next response dot redirect the user to the slash off slash signin page.

(1:48:22) Now I need to turn this into a new URL and I need to pass the request URL inside of here. So I'm going to say request dot URL. Perfect. We're just redirecting now. I don't need to return a new. Actually I'll delete that. And in here we are going to try and catch And in the try, we're going to get the data from the body.

(1:48:58) So we're going to say await request.body actually not body. Let's do it.json. Then with this we're going to get back the data. Now with this we're going to get this object that we got from here that we're going to send it. And this isn't necessarily the entire object. We need to also pass the posted by ID.

(1:49:15) So to do that, I'm going to actually just actually call directly the Prisma client. So we're going to say Prisma, which we can import by getting the Prisma object from lib/prisma. We're going to say dot job to specify that we're interacting with the job model. And we're going to say docreate.

(1:49:33) We've dealt with this before in the start of the tutorial. To create, we just have to pass a data. And in here we already have a lot of the data because we're going to get that from the JSON through the body that we're going to send. The only thing that we are not going to get is the posted by ID. So what I'm going to do is I'm just going to dstructure the data.

(1:49:51) Now I also need to pass here the posted by ID. Right? And this is going to be equal to the session do user do ID. Now when this is done, if this is successful, then I want to return the next response.json with the new job. Now let's handle an error. If there's an error, it's going to be pretty simple.

(1:50:19) We're just going to console log our console. The and we're going to say error creating job and we'll specify which error it was. Perfect. Then down here, we're just going to return a new next response. And we're going to say internal server error. We'll put a status of 500. Now, this means that we can call this um endpoint in our uh unsubmit on the post job page. So, in here, we're going to try and we're going to get a response.

(1:50:57) We're going to await fetch. We're going to fetch slappi slash jobs and we're going to pass in a method which is going to be a post method. Then we're going to pass in a headers which is going to be content type and then we're going to pass an application slashjson. Then we want to pass the body. And the body is not going to be actually inside of the headers. That's an accident.

(1:51:30) We're going to put it down here. And the body is going to be a stringified version. So JSON.stringify of our data. Perfect. Then we don't really do anything with the job response. I'm actually going to even remove that. We don't do anything with that. I'm actually going to just catch this. Get any errors? If there's an error, I'll just console log it.

(1:51:54) So, let me just console log the error. And finally, I want to pass this handle submit into the form. So, we'll go to the form over here at the top. We're going to add a onsubmit and we're going to pass the handle submit here. Now, I want to make this into a client component. So, we're going to say use client at the top.

(1:52:17) And let's try this out. Let's go to post a job. Let's create a new job. I'm going to create one and I'll be back in a second. So, as you can see, I created here a job. It's a software engineer one position at Amazon in Texas. I'm going to click on post a job. And if everything went right, we should see that information. So, it's empty. And I I I found out the issue.

(1:52:40) So, the issue is that we forgot to put in a weight right before the Prisma call. So, let me try that again. Let me try closing this up and clicking post job. And if everything worked, we should see now, yes, that data was added into our table. Also, I just realized I wrote software wrong. So, let me change that. Save change. Perfect. So, now we have successfully added that into our table.

(1:53:05) So, what I want to do as well is I want to redirect the user after this is done. So, to redirect, I actually want to come here to this and I want to after we fetch the data, I want to just redirect the user. The easiest way for us to do this is just to say window.loation.href and I'm going to redirect the user to the /j jobs route. Now I have already added here a new user. It's a consultant. And to create this user, I'm going to click post job.

(1:53:31) And if everything works out, we get redirected to the jobs page. And we should see here if I refresh the page that the consultant was added to our table. Perfect. Also, in the meantime, I added another software engineer. I just didn't show you guys but yeah. So now we have successfully created another job.

(1:53:52) Now let's set up the ability to browse jobs by showing a list of the jobs that are available and also allowing the user to search for them. So to do that we are going to come back to our page over here and we want to add a new route. Now this route is going to be on the jobs route but it's not going to be a post. So it's going to be outside of the post folder and we're going to add a page.tsx here.

(1:54:11) Now in this page let's create a default async function called jobs page. Now in here we want to return and I'm going to paste what the UI is going to look like. So this is what the UI is going to look like. It's going to have a title for the page. It's going to have an input and a select for us to uh query for the different items in the page.

(1:54:38) Also we're going to loop through the list of jobs. Now, I commented this out because we don't have yet the list of jobs. We're going to have to write an API request for that. So, I'm going to do that in a second. But you can see that we're going to loop through the list of jobs, and we're going to render each job as a card.

(1:54:55) Now, nothing really shows up. Only the search functionality shows up, but it obviously doesn't work yet. So what we want to do is we want to first add the ability for us to fetch all of the jobs available on our table so that we can actually uncomment this and loop through them. So to do that I'm going to have to go to our route.

(1:55:15) ts and I'm going to make a get request um inside of this route. So we're going to copy this whole thing over here. We're just going to paste it down here and we're going to make a get request. Now this get request is going to be different. We're going to delete most of these. Uh, we're going to leave the try though.

(1:55:33) So, let's add the try here. So, we're going to leave the try catch, but it's going to be different because there's a lot that is going to go inside of this. The user doesn't have to be authenticated to actually uh run this request. So, we don't have to get the session. But what we do want to get is the list of jobs. So, I'm going to say const jobs is equal to await.

(1:55:53) And we're going to use the Prisma client. We're going to say dot job and then dot find menu. Now if we don't put any condition here, we're going to get all the list of jobs in a random order. So what we want to do instead is we want to put a condition to order our data.

(1:56:12) So what we want to do instead is we want to add a condition to order our data. And we do that by adding an order by property over here which allows us to specify a datetime field like the posted at and say that we want to order it based on the posted at in a descending order. So we get the newest jobs first. Now when we're done getting that I'm going to return the jobs.

(1:56:35) And now if we go to the jobs page over here we can now fetch that data. Now to get the data for the jobs, we're not going to do them in the same way we did here with the post request just because I think it's going to be easier for us to just manually interact with Prisma directly on the component. This is going to be a fully uh asynchronous component.

(1:56:54) So we don't have to actually make a fetch request. We can just access Prisma directly here. So what we're going to do is inside of this component we're going to say const jobs is equal to await and we're going to await prisma.job.find many. So this is going to give us back the list of jobs.

(1:57:21) Now we can uncomment this and you'll see that when I get back the list of jobs uh I still need to comment this part out. Sorry. we we can go to the page and we will actually see the jobs being listed out. Now there isn't any particular order for them. What I want to do instead is I want to order them based on the dates in which they were posted at in descending order.

(1:57:42) To do that in the find menu, we can put here an order by and we can specify that we want to order it by the posted at field in a descending order. And now you should see that this exactly what it shows. It shows the newest jobs posting first, not the the first one to be created like it was before. Not only that, but I also want to know like the name of the user who posted this job.

(1:58:07) We have here some UI that does that, right? I just commented it out. Now, this UI gets the job posted by.name and it also gets a link, but it's just because I need to import that. That's why the error was was was saying that there was an error, but I just imported it.

(1:58:24) But where is this posted by property, right? Because if you look closely, the the type for job doesn't have a posted by. It does have a posted by ID and it does but it has a relation. If you remember correctly, it does have a relation because we created it over here. There's a posted by in a job that includes a user.

(1:58:50) Now how do I access that relation and make it such that this job will now have the posted by property which then will then consequently have the name. Well to do that to include a relation you come to the uh find menu over here and you want say you want to include and you specify which field. So I want to include the posted by. So I set it to true. And now all of a sudden we can access the posted by because now the job parameter over here include the user as part of the posted by property.

(1:59:20) So now if we go here you see that if I refresh this it should include the name of the user that posted this job which in reality it was just me. So perfect. Now we have successfully showed all the jobs. But how do we do this little thing over here? This is a little bit complicated, but I think it will introduce one of the last and really crucial parts of Prisma that I haven't mentioned yet, which is working with different queries.

(1:59:47) So it's going to be a little bit complicated if you're not going to use this every day is only when you're going to be working specifically with queries like this where I want to search my database based on some criterias. In this case, I want to first search based on name. I want to be able to write here software and then both software engineering positions are the only ones that show up.

(2:00:04) I want to maybe be able to search for only full-time or only part-time positions and or location or positions in a specific location. So the way I do this is I want to use actually this form that we have over here. If you've noticed we don't have an onsubmit to this form and we do have a button to submit it. Now what I want to happen is when I write a job here like software and then maybe I put full-time and then I don't specify a location and I press I press search. You should see that the only thing that happens is we now have this param that's changed on

(2:00:36) our screen. Right? The param now has a Q value for the query with software has a type for the fulltime, right? And it doesn't have any location. We're going to use that as the information we're going to pass to our um our Prisma query when we fetch the data. So what we want to do here is we want to get the information of what is in the params.

(2:01:02) To do that in XJS, it's it's actually pretty simple. We're going to come to the arguments over here of this the props of this component and we're going to dstructure the search params. Now, we have to give it a type and the type is going to look a bit weird, but you will understand what I'm doing. We're going to define the type as uh search params.

(2:01:23) Then in here we're going to pass a key of a string and the value will be a string or a string array or undefined. So this is what we need. Now this will also return a promise and this is something that people forget because I believe it's it only started being like this after next.js 15. So we returns a promise and to get the search param we have to say const set it equal to await search params and in here we can now get the Q the type and we can get the location those are all values that are going to be in

(2:02:10) the search program. Now what is Q? Q is query and that's why we named the the first input Q but we you could call it whatever you want. That's why we named the second one type and that's why we named the last one location. Now we want to use that to our advantage.

(2:02:29) So what we do is we say when we are finding all of the jobs we want to find the jobs where and this is where it gets a bit complicated. We want to put an intersection using an end because we are like when I search here for a specific position like software and then I also include a full-time over here and I click search.

(2:02:53) I I don't want to search for every software position and every full-time position. Now I want to search for a software position that is also fulltime. So as I introduce more of this search params I want to make an intersection of them so that we get a more concise result. So in here what I want to do is I want to put an end put an array and then we will write all of the three queries.

(2:03:18) So the first query is going to be the the Q. So if Q is equal to true then we want to implement a query for it. If not then we're just going to put an empty object. So that's just because we need to account for the fact that maybe there isn't a query here in the URL which happens right right now if I go to the page there's nothing in the URL. So that should be fine. We shouldn't just error that.

(2:03:44) So what I'm doing is if there's a Q value to the search param then I want to search for a job where the title contains the query. So if it contains that value that we're searching like in our case over here when we search software we want to check to see if it includes the word software. Now we also want to make it insensitive so that it doesn't matter if the user wrote a letter or two in caps.

(2:04:09) It it's insensitive. It doesn't like it accounts for everything. Right? Now we don't want to just search for the title. We also want to allow it to also query for information for example on the company name and on the description. So instead of just making title like this, we actually want to make this into an or and wrap this around with an array such that now we can search for queries where the title includes the query, but also that the company name includes the query. I'm going to just copy and paste it here because it's going to be the same. And I'll put the same thing for

(2:04:48) the description of the job so that it not only accounts for, for example, the description of two of the jobs include the word React, GraphQL, and Go. If I search GraphQL, I wanted to also show jobs that have that in the description, not just jobs that have that in the title.

(2:05:04) So, this is a way for us to expand our search. Now, it's giving us a bunch of errors, but that's perfectly fine. Uh the way we're going to get rid of these errors is we're going to put a comma here so that we can actually add more items since this is an array and it's an intersection. But also because maybe Q we have to cast a type to to the queries specifically right so actually what I want to do here is I want to create here a query variable. Let's see if this will fix it.

(2:05:34) I'm going to set it equal to Q, but I'm going to say as string or undefined. And then I'll just use query instead. I'm just casting this type to it because sometimes it has issues with this kind of stuff. So let's pass query query query. And yeah, that totally fixed it. It was just the type casting because we defined it like this. And when we're getting stuff from the search param, sometimes there's no type definition.

(2:06:03) So then we want to pass the next query. So now we're able to search for the user. Let's actually let's try this out. We we can search for the name of the the thing. Uh I need to put a comma here. Let's see if this is working before we add the other ones. If I click software here, let's see if the consultant one disappears.

(2:06:21) And yeah, it disappears because it gets the query for software and it keeps it. Now let's try actually searching for let me refresh this, right? And let me search for uh stuff. So stuff is in the description of consultant. So if I search that up, yeah, it detects. If I search Deote, it detects this as well. And so it's working. Perfect.

(2:06:48) Now let's add the rest of the queries. Let's allow user to query for the type. Now type is going to be a little bit simpler because it's literally just the type. So we're going to query um if type is not null then we want to query items where the type is equal to the type. If it's not null if it is null then we don't want to query anything. So literally this is all it's doing.

(2:07:15) Uh we actually have to get the type similar to what we did here with the query. We're going to call it type and I'm going to actually we need to call it um search type and we'll change this to type. Perfect. Now let's see if it works. Let me search for only contract jobs. Yeah, it's working perfectly. Now finally let's add the last one which is going to be the location.

(2:07:45) So we're going to say if location which we also have to do the same thing here. Um call this search location and we'll pass location here. If search location is not equal to null then we'll do something. If it's equal to no we'll do nothing. If it's not equal to null, then I want to search for the items in which the location contains the loca the search location that we just passed. And we want to do the mode insensitives in the same way.

(2:08:20) So that if I now come here and I search for positions in Seattle, I can write Seattle here and I'll also write it in in lowercase even though it's in caps lock over here and it will detect that this is Seattle. Perfect. Now we have successfully implemented a really cool search param inside of our uh a search functionality inside of our browse jobs page.

(2:08:44) We can search for all of these and it actually detects and get the information from our database. This is actually pretty cool. Now we pretty much have this page done. I feel like it's time for us to make the page for individual details. So when I click on this, I want to be able to be brought to a page where it just shows that specific job and it allows the user to apply for that job.

(2:09:03) So to do that, what we want to do is we want to come to our project. I'm going to create here another folder. And this folder is going to be inside of the jobs folder. And it's going to be called ID. And we're going to make it like this because then we can pass a param which is going to be the job ID as a param to that route. Then inside of here, we're going to make a page.tsx.

(2:09:28) Now I'm going to export default async function and we want to make a job page function. Now like the search param component that we just did. This is going to have a param. So params the way you do it is you do it like this. You define the params type and then you give it a type of a promise and you specify what the param is that you're going to pass.

(2:09:57) The param we're going to pass is called id and it will be a string. Then we're going to get back that param down here. So the way we're going to do that is we're going to say const job id is equal to await params do id. Then I want to get the specific job that associates itself with this param. So, with this job ID, and again, we're going to do this locally just because I don't want to deal with having to make like a a fetch request to an API route inside of a use effect because in this video, we're not using something like React Query for simplicity reasons. Uh, so I want to use I want to make the request directly

(2:10:42) here. And to do that, it's actually pretty simple. I can just say const job is equal to await prisma dot job dotfind unique because we want to find one job using one of its unique properties and the unique property we're going to use is the ID of the job.

(2:11:07) So we're going to say where and then we're saying ID is equal to params do ID uh actually we already have the jobs ID here. Sorry. So job ID then we also want to make sure that when we get the job we do the same thing we did here with the list of jobs. We include the information about the user who posted it. So we include the join that we did with the user table.

(2:11:30) So we say include posted by and we say true. Perfect. Now we want to detect if no job was found. So if no job was found, then I I honestly just want to set a not found because then this means that this ID that we have on the URL isn't a valid ID.

(2:11:56) Meaning that I'm going to call the not found function which basically triggers and forces me to be redirected to the 404 page not found uh that exists in our project. Now I'm going to paste here the return UI for this page. So this is the UI we have here. We first have a link to go back to our previous jobs. So I'll click here on one of them and we can see this.

(2:12:15) Uh so we have a link to go back to the previous jobs. We have the title, the company, the location, the price and so on and also the job description. We render all of that simply by just using the information we get back from the job we just fetched. Now one thing I want to show to you guys as well really quickly is that we use the format distance to now.

(2:12:37) What this is is a function that I just imported from date FNS and it will basically grab the date time that we got from the posted at and it's going to format it in a way that looks pretty nice like for example about 2 hours ago instead of you know just literally being a timestamp over here.

(2:12:56) So that just makes it look a lot better and we have this page. Now we want to add the ability to click on a button down here and apply for this position. So to do that, I'm going to come down here and I want to call a component we're going to create called the apply button. Now the reason why I'm not just making the UI here and I'm making a whole other component for this is because remember in order to fetch data from Prisma, we need to make this into an async component, meaning a server component.

(2:13:20) And whenever you have a button, it's by default a client functionality because it requires you to pass an on click for when you click the button. And an onclick only runs in the browser. So it has to be in a client component. So we're going to have to make a client component called the apply button. And we're going to pass the job ID to it.

(2:13:40) So we're going to pass job id. And let's create that component. I'll just create it directly over here. We're going to call it the apply button.tsx. And the first thing we want to do is turn this into a client component by saying use client. Then I want to export default function apply button. I'll pass here the job ID as a prop.

(2:14:09) And the job ID prop is going to be of type job ID which is a string. Then we'll open and close this. And the UI for this I'll manually write it. It's pretty self-explanatory. This is going to be a button. So we're going to make here a button that uh depending on that will say apply for this position or something like that.

(2:14:32) Now whenever you click on this button I want to call a function called handle apply. So we have to create that function up here const handle apply and it's going to be an async function. And when you apply for a job, the first thing we need to make sure is that we gate the ability to apply for a job if you're not signed in. So I'm going to use the use session hook that we created in the beginning of the video since this is a client component.

(2:15:00) So use session. We're going to get back over here the data which then gives us a session and it also gives us this status. Now the status will basically uh gives us the information of whether or not we are currently fetching the information about the session. So technically if the status is equal to loading what we can do is we can just return the sync button but it will say instead of being clickable actually it will be disabled because you can't click on it and it should say something like loading. while the data is fetching this it will

(2:15:39) show loading. So we can go back over here and in the handle apply function the first thing I want to do is check the session. So if no session then I want to actually be redirected. So I'm going to use the router from use router by next navigation and I want to call router.push. Now what this router.

(2:16:03) push will do is it will just redirect the user and if the user is not logged in I just want to redirect them to this sign-in page. Now if they are logged in oh also I want to return this. Now if they are logged in then I want to try to make the application work and if it fails then I need to catch any errors.

(2:16:29) Now for catch any errors I'll actually show the error. So I want to create here a state called error message and set error message. I'm going to set it equal to use state. It will be a string. So I'll make this a string type. And when you are applying uh we'll just reset the error message in case you applied once failed and then try to apply again.

(2:16:52) And then if there is an error then we're going to set the error message equal to and then we're going to actually uh let's do this. Let's check first to see if error uh the error that we get is instance uh instance of error because if it is then what I can do is I can just set it equal to the error message. So, error dot message.

(2:17:21) But if it's not, then I can just set it equal to I'll do this and I'll set it equal to set error message failed to apply for the job. Perfect. Now, what do we do in the try? Because in the try, we're going to actually uh apply the the application, right? We're going to actually add the application to the application table.

(2:17:46) Now to make that happen this time I'm going to do it similar to what we did with creating a post. We're going to create an API route that is going to handle that. So in the API/job folder we're going to create another folder which is going to be called the uh job ID folder. And then inside of here I'm going to make an apply folder.

(2:18:14) And inside of the apply folder, I'm going to make a route.ts. So that the endpoint is going to be slash jobs slash job ID or whatever you insert there slash apply. Something like that. So now in this route, what do we do? Well, we're going to copy some of the logic we did here for the for the post request cuz it's going to be very similar. We're just going to paste it over here. And uh we're going to keep it as a post.

(2:18:40) We're going to keep this session. Now from the request here, we also need to get the params. So to get the params, we're going to do the same thing we did with the components. We're going to grab the params here. Set it as a type equal to params, which is a promise. And then we have to specify what the type of our params is going to be. It's going to be job ID, which is a string.

(2:19:05) Now inside of here, we get the session. If there's no session user session ID, we redirect them. Then we need to try to find a job with a unique ID. So we get the job specifically. And then what we need to do is we need to get the job ID to try to find the job because if the job ID that is on the PM of this request is not a real job, then we want to know.

(2:19:35) So we first need to try to find a job in the job ID with that job ID. So I'm going to say await prisma.job.find. find unique and where the id is equal to and then we need to dstructure the param. So I'm going to say const uh equals to await params and we're going to get get back the job ID. So where the ID is equal to job ID and if that's the case which by the way we have to do this then if no job then we want to return uh new next response and it's going to be a job not found error which we're going to put as an error message.

(2:20:24) So we're going to say status 404. Now if the job is valid, now we know we have to also check to see if the user already applied for this job. So we're going to do const existing application is equal to await. And we're going to now make a find request to the application table.

(2:20:53) But we're going to use the find first just to see if there's any application existing already in between this job and this user then we're going to get the first one. It doesn't really matter, right? It there can't be any application uh existing if you're trying to create one. So we're going to say find first and we're going to specify that the application has to have the job ID equal to the job ID that we have on this request plus the user ID has to be equal to the session do user do ID and if that's the case so if existing application is true then we want to also return an error this error will say

(2:21:29) you already have applied for this job and it will say status 400. Now if none of that is true and we're still in this request, we want to finally create the application. So we're going to say application is equal to await prisma.lication.create. And in here we're going to pass a data which will include the job ID which we're going to obviously pass the job ID here.

(2:22:09) Then it's going to have a user ID which we're going to pass the session do user do ID and remember we also have to pass the status right and by default it's pending. I'll just pass it as well. So pending. Now when that's done, we want to return back this application as a new application created. Perfect. I'm also going to remove this console log. Now we can go back to the apply button. And in here on the try catch, I'm going to make a request.

(2:22:35) Going to get back the response. I'm going to await fetch. And we're going to make a request for the slash API slash jobs slash and I'll insert the job ID here. And we're going to pass the apply as well. Now we have to specify that this method is a post. So we say method uh is equal to post. And when we get back this response, I want to actually notify the user that it was successful.

(2:23:06) So I want to create here a state called application status and set application status. The only reason for this is just because uh I feel like there's so many errors that can happen with creating this application.

(2:23:25) There's so many times where you can fail uh that I think it's better for us to do something like this where we're going to actually make this state uh be a status of the application. So initially the status is idle meaning that the user is still applying there wasn't success or error. But this status can be either idle or success or error. And then what I can do is when we successfully fetch the data, I can set it to success.

(2:23:57) And when we got any errors, I can set it to error and so on. So also I want to refresh it actually similar to how we did here with the set error message. I want to set the application status to idle. Perfect. Now let's just update the UI to reflect this. So after applying if there's any errors so if application status is equal to error then I want to render a p tag that is going to render the error message and that's how we handle showing the error but if the application was a success then I want to replace this apply button al together so if application status is equal to success

(2:24:47) I want to make this return a div that is going to have a p tag saying application submitted successfully. And then down here I want to put a link to the view your applications page. And then I'm going to import this from next link. And the href we're going to put is the href to the last page we're going to make, which is the dashboard page.

(2:25:21) So, let's test to see if this works. I'm going to come here to the page. We have nothing on our Actually, I haven't put the UI yet. That's actually a good point. So, I'm going to copy and paste the CSS for both this and this. Uh, actually, we already put it for this, but I I'll put it for this. And I'll be back in a second. So, I just pasted here the UI.

(2:25:40) You can take a look. It's pretty simple. Also, I realized I wrote error wrong. So, I have to update that over here. And I think it should be good to go. So, let's go back to our Google Chrome. Oh. Oh, I know an error already. So, the this should not have this. So, let's go back here. Let me apply for this job. When I click on this, we should now see that our application submitted was successful.

(2:26:08) So when I click on view your applications, we should now go to the dashboard which should show the applications I've had applied. But we know that the application was successful because if I refresh this, we now see the application over here. Now let's make the dashboard. I'm going to come here to uh create another page on our project not in the API folder on the app folder. I want to make the dashboard folder.

(2:26:33) And in here let's make the page. And in here, let's make the page.tcsx. And then let's export default async function dashboard page. And I'm going to first get the session which I need to this is going to be a server component. So I can just await o. Then I want to detect to see if the session is valid. So if the session is not valid.

(2:27:06) So if user do ID, I specifically am searching for the ID just cuz I need it. I'm going to actually redirect the user uh to the O /signin page. Now this is not the redirect function I want. I don't know why this keeps uh autoimp importing. I hate that from Nex.js, but yeah. So now I'm going to paste the UI for this page and I'll go by each step of it so you guys can understand. So here it is.

(2:27:32) Now, I commented out a lot of it and I'll explain why. This is the dashboard page. If you remember from the demo, in the dashboard page, we have two different sections. We have first a section where it shows the posted jobs. So, like all of the current jobs, similar to the browse jobs section, it shows a couple of them.

(2:27:52) And then we have a section for your application. So, it shows literally your applications and like which ones you have applied and their statuses. So in this component we need to do something a little bit different where we get different types of information at once.

(2:28:11) So to do that what I want to do is actually I want to use a promise doall to make two pro to request two promises at the same time. So I'm going to say promise.all and it will return an array back with the two promises. The first one is going to be the applications. So we get back the data of which applications we've had applied for and also just the general posted jobs.

(2:28:34) Now in this promise all we also have to pass which promises we're going to have here. The first one is going to be prisma.applications and we have to import prisma. So uh let me import it. So, Prisma.lications and I want to it's actually application dot and I want to find many. Now, we're finding many because we want to get multiple of them and we want to make a very specific request for it.

(2:29:01) Now, what request do we want to make? Well, we want to get the the applications where the user ID is equal to the session do user do ID. And that's why I specifically uh gated for whether or not this is valid. Now after this I want to make sure that when I get the application I also get the information about the the job.

(2:29:32) So like if you look down here on our on our uh model right in the application model we don't have really any information about the job only when we make the relation right we can't do anything with the job uh if we just have the job ID so what we have to do is we have to say that we want to include that information so I'm going to include job and out of the job I want to specifically include the posted by.

(2:30:01) So we get the user information directly from the job information. So we're going to get that and we're going to set it to true. So we also get information about the user uh who posted that job that we applied for. So it's like a continuous like it's a it's a triangle, right? It's not a triangle. It's just a stream of of us getting the relations, right? We're in the applications. We get the job which then leads us to get the user.

(2:30:22) Now after this we have to order the applications. So I want to order it based on the apply that to get it in order of which applications we applied recently. So I'm going to set it apply that and we're going to set it in descending order. And this is pretty much the uh applications query. Now we want to make the jobs query which we have already kind of.

(2:30:51) So we're going to do prisma job.find many but we're going to do a wear and we're going to specify that it's based on the posted by ID and we're going to use the session user ID. So in our dashboard, we're only getting the jobs that we posted ourselves, but we want to use this uh cool uh special Prisma field that is is allows us to count related records.

(2:31:23) It's like not a real column in our database, but it's a virtual one that Prisma is going to add. And what that's going to allow us to get access to is whenever we get a specific job, we can now that that we created ourselves, we can now also get the count of how many people applied for this job. So to do that, we use the include property again, but then we use the special count property and we have to select what count we want to count, right? So we which relation which field we want to count and in this case we want to count applications and we're going to set it to true and uh this is

(2:31:58) going to return a number of how many people applied for this job that we created. Now below this let's also order the jobs based on the posted at and in descending order. So now if everything works fine both the applications and the posted jobs should work. Let's start with the application. So I'm going to or with the posted jobs, sorry. So I'm going to uncomment this.

(2:32:24) We also need to import the format distance to now that I already explained. But basically, we just loop through the posted jobs and we render each item individually. And then to get the count, we just say job\_count.lications. And this will give us back the count of the applications.

(2:32:41) And you can see that we now have the count for the applications. So now let's see what happens with the actual applications. We want to see the information about the applications we have applied for. So I'm going to uncomment this. And if everything works fine, uh yes, we should now see that we have the only application that we have applied for.

(2:33:06) Also, I put here links for the to post a new job to navigate there and also links for the specific jobs if you want to check it out. So yes, this is basically it. I won't do the homepage just cuz it has no value to Prisma whatsoever. I do have a homepage. I'm going to copy and paste it right now. And if you guys want to put it in your project, you can do so as well.

(2:33:26) But the actual homepage that I'm going to make uh is going to be on obviously the page.tsx. And this is going to be the code. So I'm just fetching information. It's more of what we've done already multiple times, right? But the only difference is that I'm getting the most recent jobs and I'm only taking three, which means take three allows you to specify how many of this find many you're getting back. So, we're only getting back three jobs.

(2:33:52) So, this is what the homepage will look like, right? Very simple, just reusing a lot of the stuff that we've done so far. So, this is basically it for the video. I know this is a very very projectbased video even though it's a Prisma course. But I hope the beginning the first portion of the video was definitely helpful for you guys.

(2:34:10) I hope that the rest also help you guys get more knowledge on how to actually use Prisma in a real world scenario. Uh this video was a bit different from my other videos. Usually I skip styling and CSS. I know I always do this just cuz I've gotten so much positive feedback from my subscribers saying that they love that I don't waste time on that.

(2:34:27) I only show myself writing CSS when it's a UI based video. But this video was a even more uh abstracted in the sense that I pasted I copy and pasted a bunch of the return statements. And the reason for that is because I thought that by focusing only on the parts where Prisma was being used and fetched and I briefly explained the UI stuff that I was building, it will make the course actually way more interactive, way more easy to digest without having me spending 30 minutes building some sort of form for you guys. So, I hope that you guys enjoyed this style of teaching. If you enjoyed it,

(2:35:04) please let me know what you think of this. If you want more videos like this where I don't waste a lot of time just building divs and p tags and h1 tags and so on, let me know. Uh but yeah, that's basically it. If you're interested in checking out my React course, uh it's a React course for beginner. The link will be in the description. I really appreciate if you guys check it out.

(2:35:22) Uh I put a lot of effort into that and I think you will really enjoy it. Also, thank you so much Brilliant for sponsoring this video. They really help support the channel. If you want to check them out, click the link in the description. uh cuz that's one of the easiest ways to support the channel.

(2:35:41) Just go there, check our sponsors cuz they help me be able to provide videos for you guys. So that's basically it. Really hope you guys enjoyed it and I see you guys next time. [Music]

(58) Prisma in Next.js - My Fav Way to Work with Databases (CRUD, Dev/Prod Workflow, Relations, Indexes) – YouTube

By ByteGrad

https://www.youtube.com/watch?v=QXxy8Uv1LnQ

Transcript:

(00:00) if you watch my channel you know I like using Prisma so I thought Prisma deserves its own video and specifically we're going to look at how to use Prisma in next J as the latest app router so how to use Prisma in server components server actions but also in middleware for example which runs on the edge as it's called we'll talk all about what that is serverless and Edge functions I'll explain what those are so we'll take a look at all of the Prisma main features including a typical developer workflow right so in development and then going to production how does it actually work in practice how do you do

(00:30) migrations for example don't worry if you don't even know what it is Prisma works with most of the popular databases so like postgress MySQL SQL light but also for example mongod DB now when I say Prisma Prisma actually is multiple things right so Prisma has an OM which is what we will actually use in our code this is also what will actually create the the underlying sequel for example so this video is going to be mostly focused on the Prisma omm but Prisma as a company also offers accelerate and pulse

(00:59) I'll take a look at those a little bit later so I highly recommend that you watch the video in full if you're adding a database to your nextjs app and of course today's sponsor is actually Prisma now to understand where the Prisma omm fits in in a nextjs context we have to zoom out a little bit so in a nextjs app we have both a client sign as well as a server size and actually I would say it's it's a much more server focused framework than a client focused framework right so if you're coming from the react feed world for example where pretty much everything is client side it takes some time to get used to next JS

(01:30) because the server side carries a lot of weight but this entire thing is essentially one nextjs app right but right now it doesn't have a database let's say we create a database somewhere somewhere in the world and we need to access that from our nextjs app so that's going to be from your server side we could do it directly right so we could technically directly interface with our database however in practice we actually tend to prefer using an orm like Prisma it simplifies many things and overall is a better developer experience so that's where the Prisma RM

(02:01) fits in into the overall architecture so let's actually walk through a simple practical example and I'll show you all of the main features let's say that we have a Blog and I just have a homepage right here welcome to my blog and I have a link to the SL poost route let's actually go there and let me actually show you the URL here as well so when we go to SL posts the way it works in the app router is that it's going to look for a page that matches that path right so here in the app rouder you create a folder so posts that would be part of the posts URL and then this page is

(02:31) matched with that so this page is now served here in the browser and what we have right here is just an H1 as well I just hardcoded the number zero and here I have a UR a list it's currently empty here I would like to display a list of the post but I have no posts yet so let's actually add a database and set up the Prisma omm so that we can actually create some posts into a database and then read those posts into our app for this tutorial I will use SQL light in development and post grass in production we're going to use sqlite in development cuz it just makes things a little bit easier here but if you want you can set

(03:03) up a postgress locally as well with for example Docker and follow along as well that being said Prisma has a very handy quick start which also helps you set up a sqlite database so we can essentially do anything with our database through Prisma all right so then here they talk about setting up a typescript project now when you create a next JS app with create next app that is already a typescript project right so we already have typescript installed here as well as part of the dependencies if you take a look here you can see we have tab script here already right so this is all

(03:33) just a standard simple next as boil plate that you get from create next app that we all so often do right so we already have this so if we go down a little bit what we don't have yet is this Prisma CLI so we're going to run some commands I'm opening up my terminal here and let me actually open up another one here we're going to run some Prisma commands so what we're going to do is MPX Prisma now it's a good idea to keep using the same version for this Prisma uh command so we're actually going to install that in before we even do any of that so we're going to say mpm install

(04:02) Prisma and we're going to save it as a def dependency going to install that all right so now we can initialize Prisma so now we're going to say MPX so basically we want to run Prisma and what specifically do we want to do well we want to initialize here and by default it will assume that you have a postgress database we'll take a look at that a bit later here we actually want to start off with a SQL light database it's just a little bit easier in development so you can also specify that here we can we can

(04:25) say the source of the data data source provider is going to be SQL light in our case press enter there okay so let's actually see what that gave us so now I have a Prisma folder here and here I have a schema. Prisma and so this is essentially where we have all the configuration for Prisma but also we're going to describe here how our actual database should look like like what tables or collections if you're using mongodb so we see generat a client here the Prisma client is what we use to actually work with in our code as we'll see in a second and that's it will also

(04:56) use the type it will create the actual underlying SQL queries will see that in a second and then here we have information about our data source so we said we want to have a SQL light database Prisma needs to know where it is it needs to way to connect to that well the URL for the database well we put it in an environment variable called database URL now we did not Define this environment variable but what Prisma also did as part of the initialization is create aemv file in case you didn't have it yet and if you open it up you can see that Prisma has already added this environment variable for me now

(05:25) it's SQL l so it's just going to be a file part of our file system right so that's where it put that environment variable now in nextjs we often use a different convention for environment variables we use EMV local if you actually have a secret if you want to use the environment variable as part of.

(05:45) local you cannot just copy and paste it in a different file that it's not going to work like that brma does Give an example in their documentation on how to use the EMV package to load it from a different file in case you want to do that all right so for this tutorial I'll stick to this now if you store secrets in here you probably do want to add it to the get ignore file so in the latest nextjs it should automatically add EMV but you want to double check that right so here for me it's a great out so if I push to GitHub it won't be in there all right so let's talk a little bit more about that schema this is where you're

(06:13) going to work with Prisma most of the time so we already saw the client and this data source now we want to describe how our database should look like so we have tables in the world of SQL so let's say in an application like this we want to have a table for all the post right so you would say model post and you can see it's pretty common so I get a nice suggestion here from get up co-pilot but let me actually write that out so we want each post to have an ID so you say ID and then the data type so could be an integer could be a string right for example we can say it should be an

(06:44) integer and we can tell Prisma hey this field is actually going to be the ID it has some special behavior it should be unique and things like that and we can also give it a default value so you use these add symbols we can say by default it should just Auto increment for example right so this is going to be 1 2 3 as the ID in your database this is the simplest one now the downside of this one is that if you use the ID in your application then your users May kind of figure out information about your application right so if you use it in a

(07:14) URL for example they may just change the number three with number four and so it's more likely that they gain insights from how your app is working there's also uu ID which is more complex and also a more Collision resistant version of that and let's actually use that and in that case it will actually become a string and let's say each post should also have a title right that should just be a string now when I save here watch what happens it automatically formats this file for me this will not work out of the box right so to make this work you want to do two things you want to go

(07:45) to your extensions and here I installed the Prisma extension right so here I have an extension Prisma I installed this so I think this is a must have and the second thing you want to do is you want to format things when you actually save the files so for me on Mac it's going to be command s that also does not work out of the box so if I you need to enable it in settings so if I say uh command shift p or on Windows it's going to be control shift p i want to open up my user settings here and here I have an

(08:15) entry here for Prisma right so square brackets Prisma and I'm saying here the default formatter should be Prisma Prisma here I already have editor format on Save is true for all my files in vs code but if that doesn't work you may actually also want to try adding it directly to here as well okay but for me it's working so I'll leave it like this what else should a post have well probably also actual text and some actual content right so here we can say content should also be a string so these are all strings we already saw an

(08:45) example of integer right an integer is just a number like 0 1 2 3 without decimals essentially we also have booleans here right so for example is this post already published right so here we can say Boolean and by default we can say false right now if we to create a new post we have to specify title and content if you add a default value at default it's optional if you want to make sure it's always optional regardless of if it has a default value or not you can also add a question mark there are two other fields you also often want to add so for a post we probably also want to keep track of when

(09:20) it was updated right so here updated ad is going to be some date time and actually what we can do here there is a special option for this as well is ADD updated so now press my nose Whenever there is a relevant change it should also update this field with the latest time all right and often also we want to keep track of when it was actually created in the first place in the database right so here we'll just say the default and I should just take the time at the moment of creation so because we annotated these like this we also do not have to specify any value when we create it so the actual

(09:50) mandatory values that we should specify are just title and content so we are writing this from scratch now if you already have a database and you want to use Prisma you can also introspect as it's called so Prisma also allows you to just connect the Prisma Oram to your database and then it will generate the schema for you from what's already inside the database before I show you some more advanced features we can add here as well let's actually try creating our database we don't even have a database right now remember the Prisma orm is not our database it's just sitting in between our database and our

(10:21) application right the orm helps us interface with our database but our database right now does not exist so I'm going to open up my terminal here and what you often want to do in the beginning stages of a project and you're quickly prototyping your schema because in the beginning you don't really know yet what the actual model should be right so here I typed it out because I prepared the video but in the real world of course you don't really know what you're doing in the beginning right should published should it be mandatory

(10:46) or should it be optional you don't know yet so in the beginning a command that you're often going to use is the following so here whenever you change the schema in the beginning you may want to run MPX Prisma DB push so it will sort of push the changes that you made in your schema to the actual database now here we don't have a database yet so if it's the first time it will actually create a database for us so if I say is if I press enter here you can see a bunch of things are happening because it does multiple tasks if I scroll up here

(11:16) a little bit you can see it's reading the environment variables okay the first thing it actually does here is create our SQL light database so where is the database well it's actually right here right so a sqlite database is essentially part of your file system but which makes it very easy to work with and we don't see any data in here we'll we'll have a different wave Prisma and it does some other things as well so here it also says your database is now in sync with your Prisma schema so what it actually also did is make sure that in the database we actually have a table for posts okay and then what it also did

(11:47) here is running generate so basically what it will do is it will take the post and it will generate a so-called client for us so if you've ever seen Prisma before so for example here in our page what we what we are going to do in a second is we're going to do something like Prisma dopost find many right so this is the Prisma client and right now we haven't instantiated the Prisma client yet but the Prisma client but the Prisma client will be able to give us suggestions here it will know that we have posts in our database because that client has been generated based on what

(12:20) we have here in the schema that's what generating means and to do that it did need to install one more package which it automatically did for us which is the if we scroll up here the Prisma client package right so we don't have to install this separately you could install this yourself manually but it will also be installed the first time uh DB push and I just said that it made sure to also create a table for post in our database but if I click here I can't actually see that so how can I even see what's in my database well Prisma helps you out with that as well so there's

(12:50) another command we can run here we can say MPX Prisma studio so if I press enter here it will actually open up a UI here that will show what's in the database so let me close this so what we see here is that we have one model this is going to be a table but it's currently empty right so there are no posts here right so if I click here it will say there are no rows in this table right because we haven't created anything yet but we can already see the columns here right so those columns are what we just specified here in the schema right so all of this is now a column right so we have ID title content

(13:22) so here's where we're going to have all of our post and then here in our application on/ posts we would like to show all of the posts in our database so how about we actually add one post here just some dummy data here we'll talk about seeding a database a little in a second but we can all we can already manually quickly add one here so we can click on ADD record and remember these are the only two fields that we actually have to fill out here all the other ones already have default values or are already taken care of by the Prisma Oram or the database so what we just have to

(13:51) add here is a title so we can just say first and the content the actual body of that post will be this is a wonderful okay I'm going to press save change and now you can see we have an ID and all the other ones are filled out as well and actually I maybe made a mistake here for Content I will fill it out again this is a wonderful post I will save this one change okay so now if I refresh yeah we have this one row so we have one post now in our datab how do we actually show that here on the page in our application right so the page here what we have if we take a look if I close

(14:22) this and if I open up the page here for SL poost which is this one here I just have that H1 and I have have that list which is currently empty so how do we get the post from our database or just the one post that we have how do we get that right here and actually where do we even get data in the latest nextjs app router right so if I go back to this overview one more time where do we actually even get data so traditionally maybe you had an API endpoint perhaps and then from the client you would make a fetch call to SL API SL posts right

(14:55) something like that right so in past on the client you would do something like use fact right and in here you would make some API call to your back end so this is what you would do on the front end so then on the back end on your server you have to make sure you actually exposed an API endpoint like this and make sure to connect it properly right so make sure you have to correct URL and wire it up properly so actually quite cumbersome but this is what you would do in for example a react V application now we are working in nextjs and the cool thing is in nextjs

(15:25) you can fetch data without use effect you don't even have to create a separate AP endpoint you typically don't do it in the route Handler the the place to fetch data or to get data in an xjs application is actually in the server component and so let me actually add that here so for what I call the get request whether it's with the fetch API or with your omm right we're using the Prisma omm in this case to get data this is essentially the place to do it and actually you can see I already marked this as async because what I can do here

(15:54) I can just say oh wait and then here I want to use my Prisma omm to find all the post right so we'll talk about all the options you have to actually query data with Prisma but this is what it will look like to get all of them and this will give me all the posts in my database and then I can take those posts and I can just map over them right here and so this is a server component right so here I can map over all the post and let's actually see what markup we want to render here so we want to have an Li for each one okay Li let me just tap

(16:23) through here right welcome to the age of AI programming I get a bunch of things here that I don't want but I will keep the Li and we want want to link to each post as well so we can actually view the content right very standard blog type of setup so I will actually use the buildin next link component here and I will actually tap through here very quickly the link component will actually already render an anchor tag for me so we don't need this one and let me actually import this and I'm missing a parenthesis here okay so here I'm mapping over now when I save here we're going to get an error

(16:53) because Prisma the variable here is not defined yet we have not instantiated the Prisma client here now very important Prisma has some nextjs specific documentation so for example they will show you the best practice for instantiating the Prisma client in nextjs so if you don't do it the way they recommend you may run into an issue right so here they explain as you save your files it may create another connection to the database every time you save so here they have a code snippet that I pretty much always use whenever I use Prisma I just go to this

(17:24) page you can just Google something like instantiate Prisma in xjs and you will find this article and I just copy this code snippet to instantiate that client right so what I like to do is I like to create a separate uh folder actually so let's see I will do it as part of this Source here not as app not part of the app but just a source I will create a lip folder here and then I like to just create a db.

(17:50) TS file right here and then here I will just paste that code snippet right so here what it's doing is essentially importing the client from that package that was already installed for us and then it just instantiated right here so here we have that single we don't really need to understand the exact details of this here you can see that we're assigning that here to a variable called Prisma and that is what we're exporting here and then here it's actually putting Prisma on the global object so the next time that this file runs it's already here on that global object so it will not instantiated again right so basically making sure that whenever we use Prisma it's not

(18:19) instantiated over and over again right so now I do need to import this so now my auto suggestions here should work yeah so here we see a suggestion here from lip DB right so make sure you import it from the right path okay so now let's actually see what get if I save here this all of this should work now so if I now go back here we can see I have first post here right so I added a border on the top and bottom as well but you can see now this is a very simple example of how we can interact with our database right so pretty cool that we don't need use effect or something like this we can do it

(18:48) directly in a server component this will run only on the server so we're not exposing Prisma here to the client or something like that this will run only on the server and the render result of this will be sent to the client and that's why we can still see a server component in the browser right so you don't need to create an API route Handler this is the more idiomatic way of getting data in the nextjs app router so here I'm using the find many query but if we look here the Prisma Oram also offers find First and find unique so one use case for for example find unique

(19:19) would be the following when I click on first post we are going to slash and then the ID of the post right that's what we specified here here for the link we have SL post and then we have the ID of the post so here we have SL poost and then the ID now we don't have a page file matching this route yet so if we want to have the actual content of the post that has this ID how do we do that very standard example but how do you do that in the nextjs app router so this part of the route can be a different ID

(19:52) right so this is a dynamic route as it's called so it's still sitting in SL posts right so here in SL poost we need to create another folder here and here I'm using square brackets because it's going to be dynamic now I can call it whatever I want but it makes sense to Simply call this ID and then here we have another page.

(20:11) TSX file which will be served when the URL matches this route so then here if I just copy the entire page component from here I can just paste this one and here it's just going to be post singular page we want to have some different markup right so here what I want is the following I just want to have an H1 with the title of the post and then the actual content of the post right so here I will get an error because well I need to import the Prisma client which is still going to be that same Prisma uh instance and now I'm still getting all

(20:39) of the posts here well that's not what we want here we only want to get one post and it should be the post has the ID that's in the URL so how do we even get this ID from the URL in nextjs well since we specify to nextjs with square brackets that's going to be a dynamic URL this going to be a dynamic route we can get it here in props it's called a Pam right so here we have pams in this case we only have one Pam I will ignore the type for now so pams here will have the ID so here now we can use another query here we can say find uh unique so

(21:11) here we can say we can open this up and we can say where the ID where the ID is the ID from the URL that's what we get here now what you get from the URL is going to be a string and my co-pilot actually wants to convert this into a number if we hover this you can see it should actually be a string so copilot was not helping me here but it does show here that the Prisma Oram is type save right so if you have the wrong type you will get a warning so now I can fix my mistake and now have the correct type so now it should be able to get it from the

(21:39) URL grab the correct post from the database and that's what we should be able to display here so let me actually see oh yeah so now if I save here you can see we have our post right here the title and then the actual body from the post so what we are rendering here essentially is based on what's in the URL right so very often actually you want to put information in the URL there are many benefits to that let say somebody can just copy this send it over to somebody else they can just paste that URL and when they press enter they will see the exact same also a user can

(22:12) bookmark this right the URL will be stored and then later when they come back we are rendering the exact same now here we are using the ID and technically it works but in practice we do want to have nice looking URLs so we may actually prefer to put the the title in the URL L right so instead of linking the ID right so if I close this so here on the post page here in the actual link here we are using posted ID it would be nicer to have like the title in there now if I do it like that if I now click here you can see we get some strange formatting issue here because first post

(22:48) the title of that post has a space in there so we would like to have like a dash in between and some other formatting as well so typically what people do is they create a separate field for that so if we go to our schema here so each post not only has a title but we also want to have a sort of different version of the title that is appropriate for a URL so that is often called a slug and that should also be string it's very similar to the title but it's going to be more appropriate for a URL okay so I just changed my

(23:19) schema I want every post to have a slug as well I just changed the schema for the Prisma omm but my actual database if I refresh here does not have a slug column here yet right so this this post here does not have that yet now I changed the schema I need to update the actual database again so if I open up my terminal here and I will open up another one I will keep the studio running we're going to run the same command as before right so here we again have MPX Prisma DB push if I now press enter here some of the data may have to be removed in

(23:51) order to make this work and now when I go back here and actually I do need to refresh here and go back uh here you can see see that now I also have all the data is gone but now I also have slug as a column if you don't see that after doing that by the way you may need to restart Prisma studio right I find sometimes you have to restart it it's okay so sometimes the data will be lost let's quickly add it back here I will say my first post and now for the actual slug what I actually want to use in the URL would be for example lowercase with a dash in between then the actual

(24:23) content here wonderful okay I'm going to press save changes and again I had and actually I do need to do that again for the content that's okay wonderful post press enter save change okay so now I have repopulated my database now if I go to slash poost okay I refresh here and I still see my post here also sometimes you do have to restart the dev server as well right so if you see some weird error just try restarting okay so here we are still getting all of the data from our database right that's what

(24:52) we're doing right here and now we are still linking based on the title right so now we do want to make it that slug so if I save here and now if I click here you can see we get a pretty URL here now based on the slug we now want to render this right so now if we go back to that page one more time here we get we are getting the pams and here we are still using ID so it's not going to find a post right now so here I do need to change this to slug right so ID should become Slug and actually here in the folder name I called it ID so if you

(25:26) want to get anything from the URL you need to use the name that you gave here which is still ID so we also need to change this to slug update Imports yes okay um I will save here all right so now there is one issue with Prisma because we are using find unique so you need to query by something that is actually guaranteed to be unique and if I open up the schema here the only thing here that that will be unique is actually the ID because we annotated here with at ID which by definition will be unique but everything else we have

(25:59) not specified that it's going to be unique so technically the Prisma RM is telling us hey you are using find unique and you're trying to find the post by slug but a slug is not unique here so technically from prisma's point of view there could be multiple rows in our database here that have the same Slug and I could create another row here with the same slug so therefore it's not guaranteed to be unique so you cannot query by find unique so here what you also want to add here is ADD unique which makes sense for a slug because we

(26:29) don't want to be able to render multiple different posts based on the slug we always want to have one post have a unique slug right so now if I save here I just includeed at unique here but when I use the Prisma client here I still get a warning here so the Prisma client is not immediately updated when you change the schema right so when you change the schema not only do you want to update your actual database you also want to update the Prisma client so that it has the correct types and other Behavior as well so that's why we're running this

(26:59) command every time so now I'm going to run it again right Prisma DB push will update the actual database but also the Prisma client and sometimes it will lose the data okay so I just pressed yes so it updated the database and then also generated the Prisma client so now if I go back here you can see the red quickly is gone okay now if I go to my database here the data is actually still here this time if I now refresh here I may need to restart the def server so let me restart here mpm runev if I now refresh here we can see our post back again now

(27:28) this looks a little bit better now there are many other options we have here so for example maybe a combination of these fields should be unique for example the title and content together should be unique and so then you have two at symbols there is also at map so this is actually creating camel case columns in our database if you want a different name for that snake case you can map it so then in our code we still use a camel case like this but in the actual database you will have snake case and same for the name of the table in your

(28:01) database maybe we want to give that a specific name as well you can use two ads and we can also create an index if you create an index it can improve the read performance so if you if you are often getting data by slug for example you want that to be as fast as possible it may help to create an index in case you have a lot of reads right querying by slug but not lots of updates okay so those are essentially the very Basics now here on this this page where we are getting all of the posts from our database right so here we have find many this is just blindly getting all of the posts right let's actually add one more

(28:38) uh post here here I'm manually adding it here we'll talk about how to insert data with the Prisma client in a second I can say second post going to be second- post I press enter there and then I save now we have two po now if I refresh here we see two post so it just blindly gets all of the data now we can be much more specific of course here with where we can filter that's so maybe not get all of them but only where some something is true right so maybe what co-pilot wants to do here is published where publish is true that makes sense actually but they are both false right so actually the

(29:11) default value is false so now we don't see anything right so now we're filtering all of them out right or maybe we want to have only the post well maybe something about their title so we can say title and then something specific well they should have a title okay but maybe but also other things like contains right so only the titles that contain first let's say and so if I save here you can see I now only get that one post that has first in the name right so you can get very granular here so we have contains we also have things like greater than or less than or maybe it

(29:44) should the title should end with something right so only the ones that actually end with post if I save here they are both ending with post so with where you can filter so you only get the specific data that you want now what if we want to specify the order in which which we want them so here after we filtered the data that we want and so now we know the set of posts that we want we can also specify other things about that about that resulting uh set of posts so we can say they should be ordered in a specific way right so we can use the create at field and we can

(30:15) say descending right so now you can see the order is switched the way that we're getting the data from our database right now means we're getting all of their data essentially so also their ID but also their created ad uh updated ad if you want to be a little more specific you can also say select we only want to get there all ID title and slug let's say so if I save here it still works the same because we are only using title and slug here this can be useful for example if you are actually getting a user from a database a user may have a password in the database and you don't really want

(30:47) to get the password out of the database you are only interested in other data that the user has for example their email right so then this is very helpful because now we would not get the password out the database it's a little bit saf right but let me do it like this we have a couple more options here mostly interesting for pagination right so right now we're getting all of them but maybe we only want one right so we can just say take uh one if I save here we only get one we can even say skip the first one right so we're taking one but we're also skipping the first one so now

(31:15) I get the other one again these are mostly interesting for pagination right so just as an example I hardcoded the number zero here but I would like to show how many posts I have in my database and let me actually remove this so we have two posts I would like to show two here how can I do that well the easiest way here would be to just take posts it's just a JavaScript variable and you probably know that we can just it's just an array and we can do do length on there right so it will show that there are actually two here and this works because we are actually getting all of the posts from our

(31:46) database right so we can just do posts. link now what if we have thousands of posts in that case we don't want to get all of them in that case you're going to have pages right and you're going to have 10 posts per page so in that case you're going to use take and for example Skip and so if we are on the second page and each page has 10 well we want we want to take 10 but we also want to skip the first 10 right something like that in that case this query will only give you 10 right so then we would show 10 here but we actually have thousands in our database right and about pagination

(32:17) actually they have a wonderful guide here on the website right so you can do so-called offset pagination which is what I'm showing you here but they also have cursor based um pagination right so this is a great article I recommend you check it out now here if I want to show how many posts there are in my database here we cannot just do post.

(32:34) length right because here we are only getting well 10 right so the query here will give you 10 post so if you use this variable you're just going to get 10 even though there are thousands of post in our database but we're just loading them 10 at a time so in that case you cannot use posts.

(32:53) length right so very commonly you want to get the count so here we can actually get the count with another query which is Prisma post the name of your model right this is all type save here you can see post if I mistype here Prisma will warn me and the Prisma client is all based on what you specify in in the schema when you run a command like DB push it will sort of build that into a client the types so then if we make a mistake we get a warning but we can also use do count and it will give you the total amount of posts in there I can just then I can just use that variable to show how many there are right so now we only have two

(33:28) right so I do need to remove these if I want to see them right but now uh that works as well right so when you're getting data these are essentially the options you have let me actually remove them and let's talk about updating data these are what you call the reads right reading data from your database now let's talk about writ to your database right so creating new posts updating them deleting them the data mutations fancy words and basically you have reads right reading data which what we're doing here also when I click on the second post here in the other page we

(33:58) are also reading data from our database based on the slug right this is all reading and these are essentially the main options that you're going to use but let's say we just blindly want to get all of them from our database and I will just use length this is what we have now let's talk about writing data actually inserting data into our database we were doing it manually here in Prisma Studio how do we do it programmatically with the Prisma client so let's actually add a form let's actually do it right here in the same page to to uh keep it simple so just

(34:29) added this form if I save here you will see it on the page here so we just added a simple form here just form tag with an input I gave it a type of text and a name of title and that's what we see here and then we have this text area for multiple lines it's maybe the tag you want to use and this is for the actual content and then here I have my button right create post so if I say third post here great blog post once again so if I click on create post here we would like to insert it into our database right so how do we do that now traditionally what

(35:01) you would do perhaps is you would do something like onsubmit and then you have a function here and then again you would do some kind of fetch call to let's say API post now in this case it would be a post method right and then you would grab the form data and send it as part of the body right Json stringify it and send it as part of the body right so it would be something like this and then on the back end you had to make sure that you were actually exposing an API endpoint and then you had to make sure that you have the correct URL actually properly wir that up it was

(35:33) always a little bit messy to make that work luckily these days in nextjs there is a better way actually it's to do it with a so-called server action so we can remove all of this instead of onsubmit we can use the action attribute actually and here we can specify a so-called server action so if we go back to the overview one more time we saw that in in the nexts app router to get data we can do it in Sero component now to actually update data creating updating and also deleting the idiomatic way is actually to do it in a in a server action so let

(36:05) me actually write that here as well right so your traditional post boot delete requests we can do that with a server action right we don't need to create an API endpoint and use route handlers we we simply don't use them as often anymore as before the typical use case for these actually is actually web hooks I find and there are some other use cases for them but mostly we use server components for getting data and server actions for data mutations right so here in this case we want to create something that would be your traditional

(36:34) post request so how do we do that here how do we create a server action so an action is just a function actually so a server action is just a function that runs only on the server I could technically Define it here but since these are pretty important I actually like to create a separate file for them so here as well I will create a separate folder called actions and here I can create a file actions.

(37:00) TS now here here I do need to add use server at the top this will make all the functions in here server actions and then it's just a normal JavaScript function essentially that I can export from here so here what we want to do is we want to do here we want to create a post so in here we can then use Prisma to actually create a post right so I get some nice auto complete here let's see what we get so this looks about right so this time we use Prisma post and now not find we actually want to create a new one most important part here is to specify the actual data right so here we actually have data and then here well all of the

(37:31) required Fields right so all of the fields here that are not optional that are actually mandatory required is what we need to specify here right so if I actually use autocomplete here you can see we get since the Prisma client has all of these types we can see the suggestions here right so we can see content Slug and title are all mandatory so we have title and yeah actually what should the title be well that's whatever is part of this form so how do we get access to the form data here here well if we actually specify that function here so create post I need to import it

(38:04) right so I'm importing it from that file and now react and nextjs we'll make sure that when this form is submitted the form data is actually sent from the browser to This Server action right so I get that form data right here actually right so here it's going to be a type form data so then here I can use form data.

(38:27) getet and here I need to get it by the same name that you specify here for name in the form I'm cheating a little bit here with the types because technically form data doget has a special type here for form data entry value and could also be null in the real world you do want to add some ass validation here which will also get you the correct types here but let's cheat a little bit here so we have a title and we also have a slug which is essentially going to be a derivative of the title so here what I'm going to do is just take the title and I'm going to replace the spaces with a dash and also lowercase

(38:59) everything and then here we have the content and these are the only three we need to specify you can see the red squiggling line is gone now if I make a mistake again I get a warning here right so this is all type save right so now if I submit the form all of the data here should arrive at the server side and here I'm inserting it here with Prisma so let's actually see if that works I'm going to click on create post okay so now you can see the form gets reset and now I want to see in my database if that was actually added so let's actually refresh here and you can see we have an

(39:29) entry here for the post I just submitted without creating a whole API endpoint and wiring that up I can just do it with a server action it's essentially just a JavaScript function it runs only on the server and so reacting nextjs they sort of automatically generate an API endpoint for me and they wire it up for me so I think this is actually really Innovative a lot of people talk about server components as the big innovation but I think these server actions are actually the biggest Innovation so now here if I refresh if we take a look if I refresh here this code will run again it

(40:01) will just find all of them again so if I just refresh here yeah we indeed see the third post here if I click it you can see it's all working perfectly fine right so this was a server action for creating a post now you can imagine that we would also want to edit a post right so updating in with the Prisma omm so we can have something like edit post and here we would receive again the form there would be some form right and you also need to specify which post you want to change we could do it by ID or by slug or something like that now for the

(40:33) Prisma code here it's it's just the update method here so we do need to specify which one we want to update now here it has to be a string actually right so here will be a string and then again the actual data right very similar actually all right and we can also have one for deleting a post right so we need to specify which one by ID or maybe by Slug and then we can say this post should be deleted right just one line here right we're not going to implement it here but I will leave it like that these are the main ways of doing wres with the Prisma omm in the latest nextjs

(41:03) app router now one pretty cool thing I also want to show you here if I go back to my list of posts here so here we just saw when I submit this form if I just do fourth Post Number Four you saw that when I pressed create post I had to manually refresh to get the latest data right so I had to manually refresh so it could run all of this again and actually get the latest ones now there is something else you can do in nextjs which is here in a server action for example you can also call revalidate Path so here you can essentially say I

(41:35) want to render SL poost so when it does that it should pick up on the new Post in the database and it will actually do that as part of the same request response cycle as when the data actually was sent to the server side if I now create post if I just press enter here you can see I don't even have to manually refresh but the view is automatically updated for me I think that's pretty cool as well so now now we've covered reading and writing with the Prisma Oram in the latest nextjs app router all right so let me actually close out of all of this here and let's

(42:06) talk a little bit more about relations or relationships between your models so here right now we only have posts but of course a more realistic example would be that you also have users for example right so here you can simply create another model so a user also needs an ID we also want to give them a unique email and their hashed password let's say okay so right now they are existing independently of each other but of course we can imagine that a user can have certain posts and we want to associate them together so how do you

(42:37) create these relations with Prisma so there are essentially three different relations that you could have so a very common one is a on to many relationship so a user can have many posts but every post individually can only belong to one specific user and that's a one too many relationship here so we can say a user also has post and what is that a string no it's actually well an array cuz it's multiple and not string or something like that it's actually post right so it's an array of that if I save here you can see there is actually some well

(43:08) almost magic which is that the other part that we need to specify here has automatically been generated for me by Prisma as well so here for the Post we also need to specify that there is a user that this post is connected to and we don't want to make it optional actually so we want to make it mandatory so here we have user now I actually like author more let's actually call it author we can call it author ID and then here we need we need to make an author ID as well here this author will not actually become a column in the database so this is just for Prisma so the way to

(43:39) read this is that the author ID field references the ID field on the user model right so whatever ID is here that is sort of pointing to this ID right so like a foreign key in the world of sequel so that's a one to many relation so another type of relationship is a many to many relation sh so a user may have a lot of posts an array of posts but one post May belong to multiple users right you can imagine that multiple users can collaborate on one post so a single post should be associated with multiple different users

(44:12) so what you can do is we want to have multiple authors Associated here and I don't even need to specify anything else like that I can actually just leave it like this and this is an implicit many to many relationships so Prisma behind the scenes will understand what's going on here and the other option is if you have a one to one relationship a user can only have one post so not an array but only one and one post can only be associated with one other user so in that case we can make this unique the author ID unique and here we make this optional so user can have zero or one

(44:45) post right let's make it post and a post can only be associated with one unique author ID now in this specific context I think a one to many is more realistic so a user can have multiple post post right so an array of them and here we can just leave it like this and I actually do like if the updated as and created at are at the bottom so I will move this up a little bit the order here doesn't matter you may actually also prefer putting the relations even higher but I will leave it like this all right so now

(45:14) we have changed our schema but if we go to our database if we refresh or just browse around you won't see that reflected here yet because remember we have to update our database so we can say MPX Prisma DB push and while also update the Prisma client okay we may lose some data okay let's see what we have if I now refresh I may need to restart the Prisma Studio MPX Prisma Studio yeah so now here we have two models here post and user and actually we have lost all the data we'll talk about seaing in a second so let's

(45:45) quickly add some data manually one last time so actually let's start with the user here so we only have to specify a user let's say John gmail.com and John has some hashed password okay and John currently has zero post okay I'm going to save the change here so now we have one user now if we go to if I refresh here we have one now if I go to post here let's add one post as well I'll say first post first Das post for the slug the content is a wonderful blog post okay now one last thing now is we need to associate this post with a user right

(46:18) so you can see here author and author ID are expected I can click actually here and I can pick with us right so this is the one that we want I press enter here you can see the author ID gets filled out for me as well now we can save the changes so now we have some data again now if I try all right so I'm getting an error so sometimes you may actually need to restart the dev server I find I'm just restarted the dev server NAA refresh we see our post okay and what are we doing in that page it's just getting all of the posts actually let's see SL poost we just have that one query

(46:48) from before right post at find many so just getting all of them now that we have a relationship some of your queries may actually look a little bit different so here we are still getting all of the post so here all the posts we still have it right here but what if I now want to have only the posts of a user right so here I could query my user where email is John gmail.

(47:11) com maybe I only want to display the posts of John so here this will give me the user and actually it will get an interesting error here so it says property user does not exist so the Prisma client thinks there is no user yet even though here in the schema we know that there is a user here so I'm going to make sure it's properly updated so I'm going to say DB push one more time okay it generated the Prisma client again and now actually the error is gone so that looks all right so we know that a user can have posts right so a user

(47:42) here can have post but if you query the user like this it does not automatically include the posts right so here if you do user. poost if we do this and here I do need to start the dev server again it will say post is not defined so here we you need to explicitly include the actual relation you could say so if I do this and actually here I can also make it user.

(48:07) post so now you can see I'm getting only the posts from a particular user but you need to be explicit about that now when I create a new post let's see how that would work so here I still have a form and it will still be submitted to that server action but you can see here we're already get a typescript issue because now the schema has changed so when you create a post a post needs to be connected to a particular user that's how we have defined things here post needs to have some author ID so here you say author when you create the post we need to connect it to some user so we can say

(48:39) connect this is for writing and then we can say which user well the user with email John gmail.com so now if I say second post create a post okay so that all looks good you can see this one has been created right so that's connect and that's also include here here we still have our PR quickly line by the way because it's possible that we're not going to get any user from our database right so here I actually may want to deal with a if not found a case here or just do optional chaining here okay so let me actually clean up a little bit here and let's actually start talking about seeding your database right so so

(49:16) far we've been adding a lot of manual data here initially but there is a more efficient way of doing that so it's actually with a seed script so I'm going to delete all of this data here and actually I may need to do it in particular order because they are connected so let me delete this as well the post first and then the user all right so now our database is empty again but you can imagine during development you're going to make a lot of mistakes you're going to change your database and as we saw when you update your schema sometimes you're going to lose the data so it's very cumbersome if you have to manually add data all the time so Prisma

(49:46) also helps you out with seeding your database as it's called so basically you're just going to run a script that will just insert a bunch of dummy data right so I actually like Googling this article as well because it has some handy things that we can copy so if you scroll down a little bit here here they give you an example of a seed script I'm going to copy the entire script here and what you can do is in the Prisma folder we can create a file called seed.

(50:09) TS I'm going to paste that right here here it's actually instantiating the Prisma client because we're going to run this script by itself not as part of our nextjs app it's just going to be oneof scripts that we're just going to run by itself so here we we do actually want to instantiate the client and then here we have the main function which will actually well simply insert so in the example they're using upsert which is essentially just adding but in case it already exists you can update it that's a very straightforward actually so then here the actual function is called and

(50:41) after it's done it will also disconnect right but the main function here is the meat of it and let me actually remove this because we don't want to do this we want to do something slightly different so we want to insert some posts so let's actually create an array with just some dummy data we can call it initial post and it's just going to be an array of posts so let's start off with one post so each post needs to have well a title well post post one okay and a slug right post one okay slug okay and now well

(51:12) what else do we need to specify here we don't have Auto we don't have autocomplete here no relevant suggestions because this is not typed here this array so is there maybe a way that we can use a Prisma here to help us out to type this array and the ENT is Yes actually because we have essentially described everything about the user and post there and in your application you're often going to need types for your users for users or posts right in your actual project you're going to have utility functions you're going to have intermediary functions and and places

(51:43) essentially where you need to type something in react we're going to have props for example you need to type props very often those are going to be very similar to what we have already described here so it would be kind of duplicative if we had to describe it again in types we don't have to do that luckily because Prisma also gives you types based on your models and we can use that in our application right so those types are coming from the Prisma client right so you're building the all of this into Prisma client and it has types and we can actually find those

(52:13) types so they're in here and then here under Prisma client here if I go to the index file here I can see there is actually a type for creating a post everything that is necessary to create a post there is actually a type created by pris prma for me so here I can say this array needs to be whatever that type is what needs to be an array of that right I do need to make that I don't need to import it properly here so we'll make it Prisma dots and then I will import Prisma from the client and now you can see I'm getting type safety here because

(52:43) we're saying this array needs to be of a particular shape right so now also I get auto complete I know oh yeah I need to specify content here as well and so here I'm just using a Prisma type for the seed file but you can imagine this is very useful throughout your application very often you're going to derive types from here because for a certain utility function this is not the exact type that you need maybe you want the title to actually be optional in some instance so you would slightly modify this to get the specific type you want but this is a very underrated feature of Prisma as

(53:13) well okay so here what else do we want to specify here so here we also have author so every post needs to be connected to some author but we don't have a user yet right we deleted everything so here you would do something like connect so we want to connect to a user but we don't have one yet so there's actually also connect or create so if there is not one yet we can actually just create a user here on the spot so here we can say where the email is again let say John and then we want to create right so we're going to create an actual user which is not only an email we do need email here when we do

(53:47) we do want to specify email here okay but and user also needs well let's actually use the type the suggestions here well user also needs hashed password right so let me just put some gibberish here so this is one post that's going to create a user for us we can add more posts let's actually just stick to one in the actual main function I get a good suggestion here so we start seeding and then we go over each post now the actual Prisma query to create a post is the following you see Prisma post. create and this is now a correct format because that's how we have typed

(54:20) this right so you can see there's no red Squigly lines so this should work perfectly fine so now I have my seat script so I don't have to manually create it over and over again I can just run this script currently our database is empty so I would like to run this script now how do I do that all right so we want to run this script now a lot of people make a mistake here so this is just a oneoff script that we want to run so we're not going to run this as part of our nextjs app we actually just need to use TS note actually and actually

(54:49) that same page has a nice uh command that we can just copy so here this is the command that we want to copy so actually we want to add a Prisma key to our package.json so here we can just copy this and just actually let's do it below the scripts so here we can have Prisma script so here we will say Prisma seed and we're going to use TS node to to run a typescript file one off now we do need to add compiler options here as well otherwise you're going to get an issue because we are using the es module syntax here and then this is the file that we want to run okay we do need to

(55:20) install TS node to make that work so let's install TS node as a def dependency and now to run this we can say MPX Prisma DB seed all right let's see what we get all right so that looks pretty good now if I go to my database here let's see what we have okay so now we have a user all right so we have a user and the user has posts right one post okay so this is looking pretty good if I now go to the homepage here or to/ post and refresh we see our one post and here we see the content all right so that's a way of seeding your database a

(55:50) quick note on error handling in Prisma so here when we submit the form we are invoking that server action right so here have an action create a post well actually we have other ones as well for updating and deleting and here we are using Prisma now there are certain things that can go wrong and in fact there are actually lots of things that could go wrong so you probably want to wrap this in a try catch so here I want to try to create this post in our database and if something goes wrong we will get an error will be thrown here we

(56:21) can catch that and Prisma will structure it in such a way that we can actually learn what the error is so for example here we can check if the error is an instance of let me actually import this Prisma client known request error so Prisma will have certain error codes that we can uh look up here as well right so they have a bunch of codes here so for example this particular one means that there is some unique constraint violation so here for example what we could have in our schema is that the email needs to be unique right so that's what we have here for the user the email

(56:54) needs to be unique so if a user user tries to sign up and we're attempting to insert a user into our database with the same email that already exists we will get an error here and because of this error code we know why it did not succeed right so here for example we also said that post here slug needs to be unique and slug is based on the title that's how we do it so here slug is based on the title so if I actually create a post here with the same title post one let's actually see what we get uh this is running on the server so if if I want to see the console log I need to open up my terminal here so let's

(57:30) actually submit this let's see what we get okay so now you can see we caught the error and we can deal with that as as we please also a quick note on caching caching in xjs is actually pretty complex topic which deserves its own video but just to give you some pointers here so now whenever I go here it's going to make this query to the database right so without caching whenever somebody would go here we're going to render all of this reach out to our database and send it back to user again which is not really efficient ideally we can do some caching here so

(58:00) then if the user comes to this page we don't have to reach out to our database all the way and back again so nextjs itself offers some caching so so nextjs has a so-called data cache this is actually very powerful cache if you deploy to for sale for example that data cache is also persistent even across deployments and by default if you get data with your omm it's not enabled right so if you want to enable something like that typically right so you may want to consider doing something like this here I'm using the cache function I can import this from next cache it won't

(58:32) show it for me here because as of recording it's still unstable so I need to import it as unstable cache and then I can rename it this should store the result in a data cache so so that we're not unnecessarily making database calls over and over again right so that is the data cache that nextjs offers now nextjs has another cache and in that cache you can add statically generated routes so for example the HTML of this page can be generated during the build so then we already have the HTML right so we only have to run this once we compute the

(59:03) HTML of this page and then when somebody goes here well we can just serve that HTML from a CDN right so then whenever somebody comes we're just going to serve it from a CDN over and over again so in that case we also do not have to make a database call over and over again now since I'm using prams here I'm using a dynamic route this is not statically generated by default however nextjs does offer the option to pregenerate so by default all the pages in your nextjs app are actually statically generated because that's optimal right if we if we

(59:33) can already compute the HTML during build time well we might as well do that now this route here though is dynamic right so this slug could be anything could be could technically be millions of different slugs so next CH will not pregenerate the HTML for millions of different possibilities here so if you have a dynamic route it will not be static by default however we can still pregenerate this ourselves for example the the top 100 most popular posts but it's outside the scope of this video so that's all nextjs now Prisma the company also offers something called Prisma

(1:00:04) accelerate right so this will allow you to optimize in many different ways including caching as well and so it helps you with connection pooling but also Global caching so here in the API docs they show you that you can actually also add a cache strategy to the ACT to your actual Prisma queries right so here I could then also add something like a cash strategy time to live so during 1 minute it's going to be serves from that cash right so that's also something that you may want to take a look at right so that's another level of caching that you can add as well that's something you may

(1:00:33) want to take a look at there are still some very powerful features that the Prisma RM offers like automatically generated migrations and it can run in socaled Edge environment now to show those things let me actually try pushing this to production right so there are also some production related issues that you need to know about so this is all development so this was all development and here we are using SQL light right so SQL light is essentially just a file part of your file system now if you deploy to a a host like forel SQL light

(1:01:03) is not going to work forell is a so-called serverless host so what they do essentially and I'll show you that in the dashboard as well once we deploy is that they will spin up a bunch of serverless functions to run the necessary compute so a serverless function is not a long running server right so traditionally you you had like a maybe even like a dedicated server just running in the background waiting for requests so that server is running even if you don't have any requests so it's not really efficient but with serverless functions you can run a

(1:01:33) compute as the requests are actually coming in and when there are no requests it's not going to run in the background so with serverless functions you do not have a persistent file system and therefore you cannot use sqlite here on forell and now there are other hosts like turo that actually do offer hosting for SQL light so forel does offer the a post grass option here so we can use that in production so I can go to the store tab here and I can create one database here on the free plan I can name it something like test and it's actually using the neon database under

(1:02:03) the hoods let me actually create this okay so I just created a database here on forel and here it shows you for Prisma how to connect to that so here I do need to change this in my schema now because now I'm connecting to I'm going to connect to a postgress database right so I'm going to remove my SQL light database for now and I'm just going to connect to my postr database right here okay now it does need the environment variables so here they they give me them I can copy them and I'm just going to connect to our database right here so

(1:02:33) here I can go to EMV and here I can paste my environment variables okay so now I'm connecting to my database on forel if so this could be my production database typically you don't want to connect to your production database during development but just to keep it a little bit simple here so now if I go back to my app if I refresh we will get an issue because we changed our schema and the Prisma client is not updated yet so if you try running the Prisma client will give you an error so we do need to update the Prisma client again so here I

(1:03:03) can say MPX Prisma DB push right it will not only update the Prisma client it will actually also update our actual database right and this time that's actually our production database on for sale so here you can see our database is now in sync with the Prisma schema so our database on forell where can I actually see that where is the data here well it's also here data if I now click here you can see we have these two tables that's what I just did here with DB push it connected to my for sale database and it also updated the Prisma client so now if I try going to slash posts uh I still get an error probably I

(1:03:39) just need to restart the dev server here I will restart the dev server and now if I refresh here okay so now you can see we are not getting any posts because the database that we're connecting to right now is empty right it has the tables and the and the columns but we don't have any rows and so now I have this database so I can delete that SQL light database that we had before this def.

(1:04:06) DB I'm going to delete this for now and now we have completely moved over to using that postgress database here on for sale right so in the real world what you would want to have is a database for production and then also a separate local database ideally the same type that you also have in production right so if you have post grass in production you ideally also want to have post grass in local development here right now we just have one database just for Simplicity but typically of course you don't want to connect to your production database during development so let's actually talk a little bit about migrations and a typical development and

(1:04:35) production workflow so in the beginning of a project you don't really know what your schema eventually should be you're just quickly prototyping you you're trying different things and so whenever you make a change and maybe you're maybe you're like uh post doesn't really need published right so now I changed my Prisma schema and we know that now we have to update our actual database as well right so we saw what we done what we've done so far is use one command for that which is MPX Prisma DB push and this will update our database and then

(1:05:05) published will not be a column anymore now this way we are not keeping track of the changes that we are making right so this does not produce migration files and that's fine with us because we're in the beginning of a project we're quickly prototyping and now we're like okay yeah actually it should have it should have published right so then I can save here I can quickly update my database again with this one command now over time you have a better idea of what the schema should look like and this time you actually want to keep track of the changes that you make in your schema why

(1:05:35) do you want to keep track of them well if I'm making changes in my local development database the exact same changes should also be applied to my production database right so I need to keep track of the changes I'm making so then in production or other environments or other people on my team apply the same changes so that all the databases are in sync right so now if I make a change maybe I'm going to add let's see what co-pilots can come up with for Me Maybe it has an image URL that I want to add now right so now I made a change I'm

(1:06:07) going to save here but I want to keep track of this change I just made so in development now not we're not going to run DB push we're going to say MX Prisma migrate we're going to create a migration and we have to say this is for developments if I press enter here now it will do it will do something similar to DB push but it actually create an artifact basically allowing us to keep track of the changes so then in other environments we can produce the same change okay so all data will be lost that's okay because we have a seat script as well okay so we can add a new

(1:06:40) migration so the first one we can just say first all right so then you can see this looks very similar as what we had before your database is now in sync with schema generated Prisma client but here it also says something about migrations and actually we can see that here now we also have migration here right so now here you will get a folder with all of the changes that you're making to your schema right so here is actually a SQL file and basically SQL that we can now apply right so this this would be during development we just made a change to our

(1:07:10) development database now when we push to production we have a history of the changes we made so then we can also apply these SQL files these migrations to our production database so that our production database is completely up to date as well right so that's how you would continue developing at some point when you know a little bit better about what the schema should be and you really want to keep track of changes that you're making and it's really and it's really nice that Prisma automatically generates these for you you can see when

(1:07:34) you run the migration command after it it also immediately runs the seaing files so we don't have to do that ourselves now like I just said we want to make sure that we apply these changes to our production database as well right that does not happen automatically so what we want to do is go to package.

(1:07:51) json one thing you could do you can say when we deploy when we deploy our app to forell we want to apply these changes to our production database right so here what we could do is post install we need to do two things actually but one of the things we could do is here also run that those run those migration files so to do that we say Prisma migrate and now it's not Dev it's deploy right so for production you could say and there's one other thing you also typically want to add here according to the Prisma documentation which is Prisma generate so when you deploy to forel forel will

(1:08:25) build our app now the Prisma client is in note modules we are not pushing that to forel so forel does some caching if we want to get all of the correct types also when we deploy we can run this script and it will generate the Prisma client right so this is something you typically want to add to post install all right so now let's actually try deploying to for sale and see if this all works so I'm quickly going to create a repo here on GitHub my cool blog I'm going to make it private create repository and then I'm going to push this to GitHub up I will add everything

(1:08:58) and just commit and push okay so now it's all here so now here on for sale what I can do I can add a new project click on import and now if I deploy for sale will grab will grab the repo here from GitHub and it will run a build right so it will install dependencies and then after that it will run that post install script okay so we get an issue and it is an environment variable yeah so here it's trying to connect to the database and we have our environment variables here and we have them on for sale as well but you need to connect

(1:09:28) your project so here you need to go to your project and then in the storage I can connect here and it will automatically and for sale will automatically set the environment variables for me okay now this project is connected I can just try a redeploy so here I can go to deployments and just uh redeploy here and actually the type checking is very strict so if you forget to type something it will F the build so let me actually quickly type this these param I will quickly just add that to the rebound okay let's try again all right so it succeeded let's actually open this up and let's see if we can

(1:10:00) actually work with this okay view posts and it actually already seated our database so we can see one post here if I click on this we can see the content okay let's actually try adding another one post two task create a post let's see what happens all right so now we see post two and if I click on this everything seems to be working all right the last thing I want to mention here has to do with serverless and Edge function so if I go to logs here you can see that as I I was browsing around uh certain things happened here so there was some compute here and you can see

(1:10:30) that there was a serverless function here that was servicing that request right so we just talked about servers functions so there's no long running server just a function basically more on demand and therefore we could not use SQL light but we can use the Prisma Ram now there is an even more advanced uh type of function you could say which is an edge function so an edge function is even closer to the user so that would be even faster if we could run the compute on an edge function now in nextjs by default if you have middleware so I

(1:11:01) could also add middleware to my app here I do it as a direct child of the source here so here I'm exporting a middleware function one of the main use cases for middleware is to do authentication so what you may want to do is actually getting a user from your database and you want to use the Prisma orm Prisma user and then we could try to get a user from our database so this middleware with next as is going to run on the edge other parts of your app may also run on the edge if you opt into them right so for example I can actually export a

(1:11:31) constant here I can say this needs to run on the edge right so wherever you add this the relevant compute for that will also run on the edge but the middleware by default will run there now omms generally speaking they're pretty sophisticated so they have some kind of query engine they need some resources and these Edge functions in the edge environment those are limited environments so you have more limited CPU memory you don't have access to to all of the noj S apis right so it's it's just a more limited environment and so OMS in general actually struggle a

(1:12:02) little bit with that so if you do it like this actually I will get an error here during development even and it will say Prisma Clan is not configured to run in forell Edge function so they offer you two options so traditionally you could use Prisma accelerate right so we saw it before it it can help you with caching it can help you with performance but it also helps you run the uh Prisma clients at the edge right so also on cloudflare and also helps you with a connection pool now more recently actually you can also run the Prisma Oram on the edge if you use a driver

(1:12:32) adapter so this is as of recording still in preview mode but they show you here how you can already start using that right so to run the Prisma Oram on the edge we we just need to create a we just need to instantiate the Prisma client a little bit differently right just a couple of lines like that just some just some lines like this so they basically have an adapter here based on the neon database in our case because we are deploying to forel postr which Under The Hoot is using the neon database so it would look something like this I recommend that you check out the docs for the exact code that you're going to

(1:13:03) need but this will allow you to run Prisma on the edge as well so we talked a lot about the Prisma as an omm but I recommend that you also check out their accelerate and pulse products as well so pulse is interesting because it will let you know if there is an update in your database so if you have Micro services for example there is a new user created in a database Prisma can send you a notification you may want to send an welcome email for example right so I would say check it out as well I had a great time working with Prisma and I want to thank Prisma for sponsoring this video I want to thank you for watching

(1:13:34) this video hopefully it was helpful good luck with adding a database to your nextjs application thanks for watching and I hope to see the next one bye  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(58) Full App Build - Dev to Deployment - Next.js, Prisma, Neon & Clerk – YouTube

By Traversy Media

https://www.youtube.com/watch?v=I6DCo5RwHBE

Transcript:

(00:00) hey what's going on guys so today we're going to build a full nextjs react application with a database and authentication and I reached out to two of my favorite products to see if they wanted to collaborate on this project so this video is sponsored by Neon and Clark neon is a cloud database and postgres serverless platform that I've recently used in a few projects and it's extremely powerful easy to use and affordable with a very generous free tier and it offers branching and some other cool features that I'll talk about later and then clerk is an authentication and

(00:32) user management system that's incredibly easy to use and you can have your users authenticate with just about any service including Google Facebook GitHub as well as just email and password and clerk also has a very generous free tier with up to 10,000 users monthly so there's no paying for anything or adding credit card info to to follow along with this project now as far as what we'll be building some of you may be familiar with the expense tracker application from my 20 Vanella projects course we also redid it with react and VJs now

(01:05) we're going to take that app and enhance it to use neon as our database to store users and transactions we'll also be using Prisma for migrations and interacting with the database and then we'll use uh we'll add full authentication and access control with clerk so users can log in with email and password as well as their Google account and all users will be saved in the neon database and will only be able to access their own transactions all right so we're also going to be using react's new server actions so there's no API routes

(01:38) we can just create a function in an actions file and do all of our Prisma dat database stuff from there so it's a relatively in-depth project for YouTube and I think you're going to learn a lot and you'll be able to start building your own stuff with this stack and I'll have the final repo in the description along with links to Clerk and neon so check them out and let's get started all right guys so just a couple things I want to I want to go over a couple websites and I'll have the links to all these in the description so of course we have nextjs dorg that's where you can find all the documentation for next

(02:18) we'll be using things like server actions so you might want to just use that as kind of a supplement and then I have the final repo here expense tracker nextjs so this link will also be in the description and if you need any code Snippets uh we're going to grab the CSS from here we're just using a global CSS file so we don't really have to worry about styling and stuff um but yeah so that's the repo and then clerk.

(02:43) com is what we're going to be using for authentication and user management so you will need to create an account it's there's a very generous free tier as you can see you get up to 10,000 monthly active users so very generous free tier you don't have to add credit card or anything like that and uh you'll have your dashboard with all your users so we'll be using that along with neon for our database which is a postgres serverless platform very easy to to get set up so pretty early on in the course we're going to set up our database and connect to it and then Prisma is the OM

(03:18) we'll be using for migrations for uh just interacting making queries and stuff like that so let's go ahead and just go into our terminal and just navigate to where wherever you want to create this we're going to run npx create next app at latest and then I'm going to call it expense D tracker Das nextjs and I'm going to say yes to typescript if you don't want to use it that's okay too I'm going to say no to es lint no to Tailwind because we're just using a global stylesheet Source directory no app router yes and then customize the Alias no

(04:00) all right so now it's just going to set up all of our dependencies and now what I'm going to do is just CD into that directory and then just open up vs code of course if you want to use a different editor you can all right so from now on I'm going to use my integrated terminal so I'll just close this one up and I'm just going to run the dev server so it's say npm uh run Dev so that'll open on three 3,000 let's open that up takes a second the first time you load it and there we go so now we have our just basic landing page so let's just do a little bit of cleanup so in the app folder I'm going to get rid of the page

(04:45) module CSS we don't need that and in the page. TSX which is the homepage I'm going to just delete everything and just create a brand new component now I'm using uh a specific extension called simp react Snippets and with this we can just do sfc I believe is what I what I use yeah stateless function component and it's an arrow function so let's do sfc enter and then we can give it a name let's this is the homepage so that's what I'll call it then we can tab into the props we have no props for this so we'll tab again into the return and let's put a main tag and in that I'll

(05:28) just have an H1 and we'll say expense tracker and there we go all right so the global CSS I want to change this so I'm going to grab the the global CSS from the repository the Link's in the description just go into the app folder and then Global CSS and let's just grab that and replace this with what we just copied save that and there we go all right so the next next thing I want to do is open up the layout.

(06:04) TSX and couple things I want to use the robotto font so I'm going to bring that in I'm going to select these three instances of enter with Commander control D so twice and then just change it to robotto now see how it preserved the case this this stayed uppercase this was lowercase this is uppercase R that's because of an extension that I have that I would definitely recommend to everyone it's called what is it uh right here multiple cursor case Preserve so if you're selecting multiple instances and you're changing whatever that is it'll preserve the case as the the title says

(06:38) so it's really a really simple extension but probably one of the most helpful that I have installed so I would definitely recommend that now we're getting an error here typescript error because this takes in a weight so the weight isn't optional so I'm just going to add on to this so we got our subsets and let's go right before that and let's say weight like that and for the weight I'm going to do 400 okay so that should clear up that error uh enter is not defined that's because down here we now want to change that to

(07:15) robotto and there we go all right and then we might as well change this up the title here so I'll change that to expense tracker and for the description what do I want to say for this we'll just say track your expenses and create a budget create a budget all right so that's that we can exit this file for now so now everything's pretty simplified we just have this one our homepage which obviously is the root URL um so I think that before we do anything else why don't we create our neon database so I'm going to go to neon.pdf

(08:30) schema we're just going to add a new data source for postgres and include our database URL now there's a a concept of branching with neon which is really cool it allows you to manage your database branches like you'd manage your code branches and by default we get this this main branch which is automatically created for us and branches are great for development testing and various other purposes and they improve your productivity and optimize your your continuous integration and Del delivery pipelines so a branch is a copy on right

(09:04) clone of your data and you can create a branch from uh from a current or past state for example you can create a branch that includes all the data up to the current time or up to an earlier time and a branch is isolated from its originating data so you can play around with it you can modify it and then delete it when it's no longer needed now if you want to create a a Dev Branch for example you can just click this button right here and it will create for you but since this is such a a simple application I'm going to just stick with the the main branch and if you were to go to branches up here on the left you

(09:39) can you'll see a list of them we just have the main branch I'm going to click on that and by default if we go to rolles and databases we have a role of neon DB owner and a database of neon DB so what I'm going to do is create a new role and I'm just going to call it Brad but you can call it whatever and we'll create the role uh you can copy the password or download the EnV uh we don't need to do that right now so I'm just going to close out so now we have this rooll of Brad and then under databases I'm going to create a new database and I'm going to call this uh we'll call it

(10:15) expense uncore tracker and for the owner I'm going to select that Brad role and then create okay so now we have a new role and a new database and if we go to dashboard and we want to get the connection string for that we can choose expense tracker and choose Brad for the for the role and we can just click show password as well and here's our connection string so I'm going to copy that and then let's create a EnV for our environment variables and let's say database uncore URL and then we're going to paste that in all right so we have our our database URL now a couple other

(10:57) things I want to show you we're going to be using M uh Prisma migrations to actually create our tables and stuff but you can do it from within the the dashboard here so if we go to SQL editor and we go up to right here I want to select the expense tracker database and we'll just get rid of this so since it we're using postgres right neon uses postgres so we can write SQL here so let's just experiment and create a table so I'm going to say create table transactions and you don't have to do this because I'm just going to drop it afterwards I just want to show you how this works so create a table of

(11:38) transactions and let's say we want an ID field we want it to be serial primary key okay so that should just aut autoincrement be our primary key and then we'll have a text field called text which will be the type of text I like to make that uppercase and then the amount um amount will be say varar 11 and then created uncore at that would be a timestamp and we can make that have a default of the current uncore timestamp all right so let's see if we can run this so I'll click run and you see over here we have it in the history and if we go down to tables and we select the database

(12:29) expense tracker you can see we have our ID text amount and created at and from here you can add a record so you can add it that way or we can just go to the SQL editor make sure that you have the expense tracker Chosen and then let's do insert into transactions and we want to insert the text field and the amount okay and then we'll say value vales and I'm just going to just going to go on to the next line here and we'll do we'll do a couple different things so let's say paycheck so that's the text and then for the amount we'll say 500 and then we'll put a comma here and

(13:18) let's add another one so I want to add an expense let's say gas and the way this is going to work is if it's an expense then it's going to be a negative number so we'll say - 50 right and then let's say say we have dinner so we buy dinner and that's 100 bucks so that would be100 all right so let's go ahead and run that and if we go now to tables we should see if you choose the expense tracker database we see all three transactions with the autoincrement ID the text amount and the created at now I'm not going to keep this this table so let's go back into the editor and choose expense tracker and I'm just

(14:02) going to drop the uh drop the T the table so let's say drop table transactions and let's run that and now we we got rid of that because it's we're going to recreate it through our app through Prisma migrations so now let's uh let's jump back into our application we can go ahead and make that smaller and I'm going to install Prisma so so let's open up a new terminal cuz that one's running the dev server and we'll say npm install Das uppercase D Prisma and then once we do that we want to initialize a schema okay so it's installed now we can

(14:47) run npx Prisma and a knit okay so by doing that it should create a this Prisma folder and inside of it we have a schema. Prisma just make that a little smaller all right so we have our data source DB postgres we're looking at a a u a environment variable called database URL which we are we have right here that's our neon database so that's that's all set up now what we want to do is create our model so let's go down here and basically we're going to have two models one for users and one for for uh transactions and these models

(15:30) are going to be used with our migrations to create the actual database tables so we want to create all of our Fields here so let's say model user and the first field we'll have is an ID which will be of the type string I want it to be the primary key so we do that with at ID and then I want a default value of a uu ID so we can do that by passing in the uuid function okay and uuid stands for uh I forget what it is it's Universal Universal something identifier I forget um but yeah so that's going to be the the regular ID now in addition to that since we're using clerk we're going

(16:13) to the user is going to be created within Clerk and it will have its own ID there as well so I want to store that in the database too so let's call that clerk user ID okay now you can have clerk authentication without doing this without having the user store in your own database but we want that user information in our database we want the user's email name and stuff like that so that's going to be a string as well and I'm also going to just add at unique because obviously we don't want two users with the same clerk user ID then we're going to have the email so that's

(16:50) going to be a string and that's also going to be unique then we'll have the name and say string now name I'm going to say optional so we'll add a a question mark there and then image URL because when you use clerk you can have a an image so string oops that should be uppercase okay so name image URL let's do created at and that will be the type of date time and let's add a default so we'll say default and then we'll pass in now and let's do updated at that will also be a date time and we'll say at updated at and then we're going to have a relationship between

(17:44) models and transactions because when you log in you obviously you only want to see your transactions so they have we have to be linked so we're going to say transaction or transactions and we're going to set that to the type of transaction and it's going to be an array so we want our brackets and right now it's going to have a red line because transaction doesn't exist so let's do that next we'll create the transaction model so model transaction and then here we want again an ID it's going to be a string it's

(18:21) going to be primary key and we want a default of a uu ID and then we're going to have text so text will be string let's do amount and amount we'll store as float uh what else and then for the the relation to the user let say relation to user we're going to have the user ID for that transaction which is going to be string and then we're going to have a user field and and this is where we want to create our relations so we're going to say user and then user with uppercase and then at relation and then our Fields so we pass in here

(19:11) fields and in Brackets user ID and then we want references and the reference we want to use is clerk user ID and then we can set on delete because if you delete a user then you're going to have transactions that belong to a user that doesn't exist so we want to make sure that we Cascade the delet so that if you do delete um a user then all of their transactions also get deleted all right and then after that we're going to just have a created at and that's going to be date time and we'll do a default of

(19:55) now and then we're going to add an index of for the user ID so we do double at signs and then index and then user ID all right so that should do it as far as our model and when we run our migration it should create the the table tables for us so we don't have to manually create them like we used to have to do back in the day with SQL databases okay so let's um so we modified the schema now we want to generate the client so we can do that let's jump into our terminal here and we're going to run npx Prisma close that up Prisma generate so whenever you

(20:40) Mo if you modify this if you add or change something you want to run this npx Prisma generate so Prisma schema loaded okay now we want to create our migration and run the migration so we can do npx and and then Prisma migrate Dev and that should create the the tables in neon okay enter a name for the migration let's just say user uh user transaction create now it says your database is now in sync with your schema now before you look at the database you'll see that a migrations folder was created and if we

(21:37) look at this file right here this is what was run create table user and it has all the fields that we have in our model and then create table transaction all the fields there any indexes any foreign keys so any relationships between you know the users and the transactions so that should have been run now let's check Neon so let's go to tables and if we choose our expense tracker database there we go we have our migrations and then we have this transaction which has the fields that we have in our model the including the user

(22:16) ID and then the user has all the fields that we have for our user model so our database is now set up and and ready to use so the only thing other thing I want to do before we move on to to set up Clerk and set up authentication and all that is create a file that we can use to interact with our database a file that will initialize the Prisma client so we're going to put that in the root we're going to have a folder called lib folder called lib Li and then in that we're going to have a file called db.

(22:54) TS okay so here we can import the Prisma client okay and that's going to be from not that it's just Prisma Prisma Cent okay and then I'm going to declare a I'm going to say declare Global and V Prisma Prisma client or undefined so what I'm going to do here is kind of like a hack because nextjs has Hut reloading and we don't want the we don't want to initialize pris the Prisma client over and over so we have to kind of add this little hack here so we're going to say export const DB because we want this DB

(23:45) variable to be exported because that's what we're going to bring into our into our other files where we want to use the database and we're going to set that to let's say Global this do Prisma or new Prisma client okay and then we're just going to check to see if we're not in production because in production we just want to initialize it normally but if we're not in production then we want to uh we want to make sure that we don't initialize it multiple times with nextjs hot reloading so let's say process. env. node EnV and we're going to check if it's not equal to to

(24:31) production and if not then we're going to say Global this. Prisma and set that to our DB okay so that's it in this file we don't have to touch at all but we can now bring in this DB variable and we can use it to to make queries whether we're adding something or updating fetching whatever it is we're doing so that's our main database file all right so now I want to start to set up clerk our authentication all right I know we haven't really done anything in our app yet but setting this stuff up first is going to make it easy for us to just to

(25:11) just create our components and do what we need to do uh I always I always think that well I shouldn't say always but in most cases I think it's good to just get your database stuff set up get your your your API keys and and all that that you need get that set up before you start building components so now let's install clerk so I'm going to go back to let's go down here clear that up and let's say npm install and it's going to be at clerk sln nextjs that's the package we want to install and then we're going to go to the clerk dashboard okay so we're going to sign in

(25:53) and everything here I'm doing you can do absolutely free so I'm going to sign in with GitHub and um I have this expense tracker Dev I don't want to use that so you're going to want to create a new application and over here is where well first off let's give it a name I'm just going to call it expense uncore tracker and then you can choose these different methods of of logging in so I want the defaults which are Google and email and password but you can also use GitHub Microsoft Dropbox all kinds of stuff here but that's all we're going to need so and our our login will look like this so

(26:35) let's click create application all right now what we want to do is let's see so we already installed it we already did this and then we need to set our environment variables which is going to be the public publishable key and the secret key so I'm going to click the copied I'm going to click the um clipboard thingy here and then let's let's go into ourv and paste that in so obviously yours will be different don't use mine mine won't even exist by the time you watch this but this should be your publishable key and secret key so let's save that close that up and then we it

(27:15) says update middleware TS so update your middleware file or create one at the root of your project and yeah so we want to copy that and go into the root here and we're going to create a file called middleware dots and then just paste that in so this will initialize the the clerk middleware and then the next thing is to add the provider to our app so we want that to go in the layout so if we want to use clerk we have to wrap everything in the provider so let's import that up here let's say import clerk provider and that's going

(27:57) to be from clerk SL nextjs and then we're going to put that in the return we're going to wrap everything in that clerk provider okay so let's take this ending tag and let's put that down here just like that and we should be all all set to to use clerk authentication that it's as easy as that to get set up so now why don't we create our header component because that's where I want to have the the login button so so uh let's see do we even have a components folder yet no we don't so let's create a folder called

(28:35) components and then a file called header. TSX we'll say sfc so we're going to create a component called header and let's see in the return let's have a nav tag and I'm going to give it a class of navb now these classes are in the global CSS file that I've added so we're not going to have to worry about any writing any CSS and then we're going to have another div in that called navbar Dash container and this is you're going to see just how easy this is to have a signin button and sign out button so and and some options for settings so really

(29:19) cool stuff so let's have an just an H2 for our our logo thing so we'll just say expense tracker and then then we'll have a div so this will be on the other side and again it's already styled so basically I want to I want to import a couple things from clerk so let's import and this is going to be from and then at clerk sljs so what I want to bring in is going to be the sign in button and then signed in which we'll check to see if we're signed in and then then signed out and then user button and the user button is basically the little Avatar thing that you can click on to to you know log out and see your settings

(30:10) and so on all right so now in this div I'm going to use the signed out component so we want signed out and whatever I put in here is going to show if we're signed out if we're not logged in so what I want to show is the signin button all right then underneath that let's say if the user is signed in then let's show the user button all right so we'll save that and let's go back to our thing here and we have to obviously bring the header in now the header is going to go come into the layout because obviously we want this on every page so let's import the

(30:56) header from component at component slhe header and then we're going to put this let's see we have body let's go into the body and let's wrap this this children I'm going to have a main tag and actually I want Main and a class of container okay and then we'll move this children into the main and then above the main we'll have our head header oops header component okay so there we go so we have a signin button right I'm going to click on that and it's going to show us the sign in and I'm going to log in with one of

(31:45) my Google accounts here I'll use this Tech gu info click continue and that's it we're now logged in and you see the user button if I click that we have this drop down where we can sign out we have this manage account where you can update your profile you see your email address you can add additional email addresses you see your connected Google accounts so that was like nothing to have this functionality and to do this from scratch on your own would be a lot of work so really cool stuff so if I sign out it just signs me out and now I see

(32:28) my signin button again now the issue here is well not really an issue but it's not saving our user to our database it's clerk is working right that's all set up and you can you can add you know checks if you want to show something if the users logged in or whatever but we want to save that user in the database okay and doing that is is pretty simple and there's a lot of different ways to do this so initially I was going to use something called Web hooks to run a function when a user is created because you can do that and that takes some extra work because you need

(33:03) to make your app publicly accessible so when we're working locally like we are right now we would need to use something like enro and this is initially what I was going to do but I ran into some issues where I had to deal with uh opening ports on my router and I don't want you guys to have to do that I don't want to have a project that you know a quarter of the way through you can't continue because you have some firewall issue or whatever um so some things are just not great for tutorials even if even if that's the more common way to do it um but what we're going to do is something very simple we're just going

(33:39) to create a utility called check user and it's going to get the current user you know the logged in user the clerk user and see if that user is already in the database if it isn't then it's just going to add that user to the database okay and it's just going to return the user if it's already there it'll just get it and return it so let's put that in the lib folder so we going to create a file here we're going to call it check user. TS all right and there's a couple ways that we can do this with clerk so I'm

(34:12) going to import this current user from clerk nextjs server and then we're going to also import our database remember that that db. TS file that's where we're going to use that well we're going to use it in a lot of different places but we're going to use it right here so let's say at SL lib slash and then DB okay so that's our database file and then we'll make this a function uh we want to export it too so let's say export const and we'll call this check user okay so we'll just make this an an async arrow

(34:52) function all right and then let's let's get the user so we can say cons user and set that to await so this is asynchronous so that's why we're using a sync await here and we want to get the current user all right then we want to check let's say check for current current logged in clerk user so we'll say if and then if not user then let's return null okay so if there's no user then we'll return null um then we want to check if and what I mean by that is it's checking clerk not checking our our database but we want to check if the

(35:47) user is already in the database so we can do that by let's say const log pluged in user and then we're going to await and then this is where we use our DB file so db. user Dot and then I'm going to use find unique okay so these are methods that pris uh Prisma offers so find unique is going to take in a where Clause so we can say where and we want to compare the clerk user ID is that equal to the user.

(36:26) ID all right so that will tell us if the user is in the database and it will get it for it will get the user so now let's go underneath that and let's say if user or if user is in database then return user so if logged in user then we'll return logged in user and then let's say if not in database create the new user all right so for that let's say const and we'll just say new user and let's set that to await and then we're going to use our db. user.

(37:15) create okay and that's going to take in a data object and what we want to add here is clerk user ID we're going to set to user. ID so user is the user that we get from clerk right so that's right here getting it from Clerk and we want to save that user ID as the clerk user ID and then in addition to that we want the name now what you get back from clerk is going to be you have a first name and a last name you could have first name and last name in your database as well but we just have just a name so what I'll do is add some back

(37:54) ticks and we'll take the user. first name and then just a space and then we'll get the user dot uh user. last name okay so that will add the name to the database then the image URL so clerk also gives us an image URL field and then email is going to be user so what we get back here is this user email addresses because there can be multiple email addresses so I'm going to get the first one so it's the zero index and then email address and that's it so that should create a new user for us and then we just want to down here

(38:41) return new user and that should do it so just to reiterate we're getting the current user from clerk we're getting our database file we have a function called check user and then and we're exporting it we're getting the user if there's no user like if we're not logged in as I'm I am right now then it's going to return null uh then here we're checking if the user is in the database right because obviously if we pass this point there is a user so we're checking if it's in the database using the wear Clause if it is

(39:18) we're just going to return the user if not we're going to create the new user with the fields from Clerk and then we're going to return the new user all right so now what we'll do is go into the header and the header is loaded on every page right it's in the layout so even though this is just a one-page project but even if there were multiple pages so let's bring in check user uh yeah so bring in check user from lib check user and then all I'm going to do is in the header I'm going to make this async

(39:59) okay this is technically a server component runs on the server so let's say cons user and set that to await check user so that's going to run it all right so I'm going to go ahead and save that and just to make sure that that's actually running let's come up here and let's console log user and we'll save that and remember this is a server component so it's going to show in this console which you can see right there null if I reload there and it also shows it in my editor I think that's the turbo console

(40:37) extension that I'm using that does that um but yeah so that's null obviously because we're not logged in so let's click sign in and I'm going to just use Google I'll use this Tech Guy info and what should happen is it'll log me in just like before but now it's going to get saved to the database and you can see my console log now is this so this is the stuff that I get back from clerk the the ID updated at image URL has image primary email address so all this stuff you have access to this is the email addresses object

(41:16) um what else phone numbers so you get a ton of stuff here to work with now for the moment of truth let's see if it was actually added to the database so we're going to go back to Neon and user I'm going to reload the page and make sure I choose expense tracker and there we go so we have all the user data we have the ID so that's just the our you know our database ID the clerk user ID the email the name the image URL created at and updated at okay so anytime a user signs in in with either Google or through email you know

(42:00) they sign up with email and password it's going to get added to the database and that new user file a check user file is going to always check to see if the user is logged in or not and it'll get the user so I'm just going to get rid of that console log so yeah I mean with the little bit the little amount of code that we've written we have some really cool functionality okay now we're gonna just basically just show a message with a signin button if the user is not logged in so let's go ahead and log out sign out and I'm just going to do this in a

(42:38) very simple way we're just going to have a component called guest. TSX and let's just do let's see we're going to do sfc and we'll call this guest and then as far as what we want to return let's just have a a div with the class name of guest and we'll have an H1 we say welcome we'll have a paragraph and we'll say please please sign in to manage your expenses or let's say manage your transactions and then we can just add the signin button by bringing it in so sign uh what is it sign in button from Clerk and then we'll just put that right under the paragraph So sign in

(43:34) button all right so now let's Go's see let's go to our homepage because that's where we want to show it so it's this page. TSX and let's bring in say import guest and then I'm also going to import the current user and let's go uh this homepage component let's make this a sync and then we'll say con user set that to await current user and then all I'm going to do is before this return I'm going to check to see if not user and if not user then let's return guest so ve very simple way of doing it and then if if there is a user let's change this this uh expense tracker

(44:32) heading here instead of that let's say welcome and we should have access to the user's name so user is coming from clerk so we have access to first name so let's just do that for now and now we can click either of these buttons to sign in just going to hit that and techi info continue and there we go welcome Brad okay so that's that now what's the next thing now we can start to work on our transaction components we have everything set up as far as basic authentication as far as the database goes and our models and tables so let's

(45:13) start off with the add transaction component because I want to be able to have a form where we can add our transaction and get that sent to the database so let's create in components a new file called add transaction. TSX and we'll do sfc add transaction and then in the return here let's just add a fragment and let's do an H we'll do an H3 add transaction and then we're going to have our form now our form we're going to be using server actions however I don't want this to go directly to a server action

(46:03) because uh I want to do some validation and in order to do that we're going to have it go to a client action because you can now have actions on your client as well and this is going to be a client component in fact we have to add to the top use client like that or not like that like that okay so what I'm going to do is have the action go to a function called client uh client action like that and we'll add this so we don't get an error we'll go right above the return and let's say const client action and we'll set that to an arrow function and for now let's just do

(46:55) const uh not const console I'm just going to do a console log and I should be able to get the form data uh wait a minute we need to do a couple things here we need to first of all this needs to be async and then it's going to take in form data which is going to have a type of form data uppercase F and d and then in the console log let's do form data doget and we're going to get the text field and then I'm just going to put a comma and then we'll do form data.

(47:33) get and we want to get the amount okay because we can do this now because of these actions which is really cool and then we'll call our server action from here because obviously when we when we add the transaction we're going to be using Prisma and that's that's obviously backend stuff you're not going to do that within the client but there is some stuff I do want to do in the client that's why I'm having it go here first all right so now in the form let's have a class of form control and let's have a label and this

(48:05) is going to be for the text so we'll say text and let's see have an input and let's give that an ID of text a type of text and also a name of text because that's how that's how we're getting it here this form. datag get is going to look at the name name of the input fields and then we'll have a placeholder as well and just say enter uh we'll just say enter text dot dot dot okay now let's go under that div and add another form control div and then we'll have another label let's close uh close a sidebar up so this is going to be for amount and in the label

(48:50) for amount we're going to say amount and then I'm going to put a line break and then I just want to put in parth es that negative uh so negative is an expense and positive is income just to let the user know and then input let's do a type of number and a name of amount and ID of amount and a placeholder of enter amount and then also if you if you have a number field it's not going to allow decimals by default but you can change that by adding a step of 0.

(49:43) 01 uh no zero yeah Z 0.01 and then it should be able to have um decimals okay so under that last div let's add a button give it a class of BTN and we'll we'll just say add transaction all right so let's bring this in now to our P our homepage so app page.

(50:12) TSX we're going to import add transaction and we're going to put this right below the H1 add transaction and there's our component okay we got our text field we got our amount we should be able to do like $4.99 cool now if I submit this and I look in um oh this is on the client that's right remember we said use client and you can see right here uh s SWS which is the text in 499 so this is a client component however we want to submit it to a server action so let's create that let's go to app add a new folder called actions and we'll add a new file and let's call this what do I want to call this uh we'll

(51:03) just call it add transactions and that's going to be a TS file or add transaction so singular so add transaction. TS now with the server action we're going to put at the top here use server instead of use client okay so this is this is only going to run on the server and then let's we're going to have a function called add transaction and it's going to return a promise with a transaction result so since we're using typescript let's add an interface so we'll create an interface I'm going to call this transaction data and this is going to have text

(51:53) which is going to be a string and it's going to have amount which is going to be a number all right then we're going to have another interface call and if you're not using Tri typescript you don't have to do this but let's say transaction result and the result is going to be either it's going to have data and I'm going to put a question mark because it could be an error but what we what we are expecting is data and that will have the type of transaction data dat okay then we could have an error as well

(52:30) again we want to make that optional so question mark and that would be a string so this function that we're about to create is going to return either the data which will have the transaction data or an error which will be a string and we're returning the error because we want to be able to show it in the UI if there is an error so now let's create the function so we're going to say async function add transaction and it's going to take in form data so let's say form data with the type of form data and then what it's going to return is a promise and that

(53:10) promise is going to have a type of transaction result now I want to get the data from the form data and check I want to do a little bit of error checking or validation and then send back either the transaction if everything's okay or the error if there's an error we're not going to do anything to do with the user yet or anything to do with the database I just want to get that part of this working so let's create a variable we'll call this uh let's say text value uppercase V value and let's set that to

(53:50) form form data. getet and then text okay and then we'll do the same for amount so let's say amount value and that's going to be form data. getet amount all right then we're going to let's say check for input values so I'm going to say if uh if not text value or the text value is equal to nothing just an empty string or not amount value value okay so if if that if any of those are true then what we want to return is going to be an error which will be a string and we'll say text or amount is missing okay because remember we can return from this function either uh either error or data so in

(54:57) this case we're returning error and then next thing I'm going to do is ensure that the text is a string so let's say const text and type that as string and then we're going to say text value and then do2 string okay so that will ensure ensure text is a string and then let's do const amount and we want amount to be number so let's set that and we're going to use the parse float function and then pass in amount value.

(55:49) to string okay so we'll say parse amount as number okay so we're just formatting it and making sure it's how it's supposed to be before it goes in the database now let's create uh an object called transaction data oops lowercase T lowercase T because we want to set that to the type of transaction data uppercase t set that to an object which is going to be text and amount all right and then the last thing I want to do here is just return so as long as there's no errors then we're going to return that data which is going to be our transaction data so that should satisfy everything

(56:34) as far as the promise that we're returning with the transaction result and so on now we want to make sure we export this function so down at the bottom I'm going to export it as default add transaction now let's go back to the add transaction component and go into the client action cuz that's where we want to call the function we just created we don't want this console log anymore what we want to do is create a variable to hold the result of the server action so we're going to set that to uh let's set that to await because

(57:10) remember it returns a promise add transaction which I'm we have to import which it just it auto imported for us sorry about that Discord so we're importing add transaction and then we're setting it to the uh that result variable and we want to pass in here remember it takes in if we look right here it takes in form data so we're just going to get the form data here and then pass it in like that okay so that will give us a result and that result is going to be either data or error remember that's the transaction result it's going to be

(57:49) either data or error so let's check for the error so we'll say if the result. error we could do it like this or we could just uh we could just destructure this and say data error and let's say if error either way it doesn't matter whatever you want to do so if error then let's alert and I'm going to not going to keep the alert we're going to use a toast and make it look better but just for now we're going to use an alert and then else we'll say say alert and just say transaction added even though it's

(58:30) really not we haven't done any database stuff yet but we'll just do that and then I also want to just console log the data okay so at least we can just try this out so far I'm just going to reload this okay and let's say like uh paycheck paycheck 400 and add so we get trans action added and if we look at the console log we get text paycheck and amount 400 all right so again what happened is the form submitted to this client action but then from there we called add transaction passed in the form data got back a

(59:15) promise and got this data or error in this case no error so it just alerted transaction added and then console log the data okay okay if I try to submit this without anything then I get text or amount is missing right and yeah we just got it's just an alert of the error so now before we um before we add it to the database we need to get our user because we need to uh add the user ID with the transaction so it knows which user added that transaction now within our AC s to get the to get the logged in user we're going to import off from clerk nextjs server

(1:00:07) we're going to do this a little bit different than um than current user this was actually suggested to me by the clerk devs so down here uh let's see where do we want to do this let's just go under all this validation stuff and let's say check uh or let's say get logged in user and we can actually extract the user ID from that off okay so user ID from off and we can try it out just to make sure oops just to make sure we're actually getting it let's say user ID and this should log down here because this is a server server action

(1:00:56) so we'll just put whatever okay transaction added and then down here you can see our user ID so we want to make sure that that user ID goes into the database when we do our our dat our Prisma call all right um but first we want to check for the user and if there isn't a user then we want to return an error so right under that let's say check for user and we'll say if no user ID then let's return error because remember we can return an error which is a string and we'll say user not found okay and that should do it as far as getting the user now we want to actually before we do the

(1:01:47) database why don't we get the toast set up instead of doing these alerts so I'm going to go to the the other tab here in my terminal and let's npm install react Das toasttify and if you've watched in any of my other react tutorials chances are you've seen me use this before it just gives us nice looking uh call outs or nice looking toast components instead of having an ugly alert now in order to use this in our layout we have to Output the toast container so let's go to layout TSX and at the very top we're going to import two things one we need the toast

(1:02:29) container from react toasttify and we also need the CSS file which is going to be uh react Dash toastify slist slash and then react I believe it's yeah react toasttify docss okay and then we need to Output this this toast container down here in our layout output so we can put it just right above the ending body tag so toast container like that now from any component we should be able to import toast we want to be in the component not the uh not the the server action so here let's import and like that so import

(1:03:23) toast from whoa from react ah react Dash toasttify and then let's see where we want to call this is going to be right here instead of alert let's do toast. error and then pass in that error and we can just get rid of the alert here this is ultimately um we're just going to redirect or or refresh but for now we'll just leave that I just want to make sure if we get an error there we go texture amount is missing so that looks much nicer than the alert all right so now the final thing we want to do for this component this whole part of the application is is save it to the

(1:04:14) database so we're going to do that in the server obviously so add transaction TS and let's see we're going to import a couple things here one is going to be the DB that's what we whenever we want to use the database we're going to bring in that file and then we're going to bring in something else called revalidate path because once we actually um once we actually submit the transaction we basically need to to the app to refresh and this is going to do that for us this revalidate path it's going to be from next cache okay so now let's go down to where

(1:04:54) we have this transaction data and let's see I'm actually going to change this now to equal get rid of that so let's have this equal a weight and then DB do transaction and then do create and that create is going to take in an object with a data object and that data object will have the text so it'll have the text it'll have the amount and the user ID all right and then actually you know what we should put this in a try catch so let's do this try oops try catch and I'm going to move this and the return just going to cut that and put that into the

(1:05:52) try all right so now we have transaction data now in between these two is where I want to call revalidate Path and then just pass in pass in slash so that way we don't have to refresh the page once it gets added to the database and then if there's a problem so in the catch then we'll simply return uh we'll return an error which will be a string and we'll just say transaction transaction not added all right all right so I think that should do it so again it's just getting the form data passed in we're getting the text and amount validating

(1:06:34) it a little bit we're checking uh making it making sure it's a string and a number getting the user ID creating the transaction with the text amount user ID and then we're returning the transaction data now let's go to the component and I'm just going to have a toast let's say toast. success s and we'll say transaction uh yeah transaction added okay so let's try that out and hopefully it works if it works it's going to go into the database so let's say paycheck 500 add we get transaction added we obviously can't see it in the application yet because we haven't added that component but we will have a list of transactions but let's go ahead and

(1:07:29) check the database so I'm going to reload the page here and choose expense tracker and we have our user we know that already and there's our transaction it has its own ID it has the text of paycheck amount 500 it knows what user added it and created at awesome so that's working as expected now one thing I would like to do is clear the form so we can do that by using use ref so I'm just going to go up to the very top here and just say import and use ref which is just a react hook so that's going to be from react and then let's go right in the add

(1:08:15) transaction and we'll say const I'll call this form ref set that to use ref and I'm just going to add here for typescript HTM ml form element and null and then what we can do is we need to attach this form ref to the form so right here let's say ref set that to form ref and then just right after the success we can take that form ref and we can say dot current and then do reset okay so that should clear it so let's add let's say dinner we'll say negative 50 let's try a decimal so we'll say negative U 4998 add transaction okay then we're going to just go check

(1:09:18) it reload expense tracker and there it is dinner and it has the negative 4998 and it has our user cool so our whole add transaction functionality should be complete so the next thing I'd like to do is show the balance right so we'll have a balance component but this is going to be the flow of of the rest of our application creating the component creating the action whether it's an action of well we have no more forms that'll be submitted so it won't be a form action but for instance we get the user balance we'll have an action that will reach out to the database with Prisma get that

(1:10:03) balance and then send it to the component so we can close all these tabs up and we're going to create a new component and we'll call this we'll just call it balance. TSX and in Balance where this is going to be pretty simple so let's just do sfc and call call it balance and let's see in the return we're going to have just a fragment with an H4 and we'll say your balance and then under that will be an H1 and this is where we want to put the balance but for now I'm just going to put I'm just going to hard code 500 because I just want to bring it in so

(1:10:49) we're going to bring it into page. TSX the homepage and let's go right under add transaction and we'll import balance and let's go down here and I'm just going to change the H1 to an H2 and let's put the balance right under the H2 and above add transaction okay so there we go your balance 500 now we want to get that balance from the database and we can do that by using a simple reduce method on the transactions so let's go into actions create a new file here called what do we want to call this get we'll say get user balance. TS and then from

(1:11:39) here we want to mark this as use server and then we want to bring in our database so import DB from lib DB and we're also going to import off and that's going to be from clerk next JS server oops all right now let's create a function so we'll say async function and I'm going to call this get user balance and then as far as what we're going to get from this it's going to be a promise and it'll be an object with the balance which will be a number and we can also get an error which will be a string all right uh whoops I forgot my

(1:12:32) quotes not my quotes my what is this thing colon okay so in the function here let's get the the user ID remember we can get that from off so we can destructure and get user ID from off and then we'll check for the user so if not user ID then let's return an error and we'll say user not found all right and then if there is a user then we're going to open up a TR catch and in the TR catch I want to first get the transactions of the user so let's say transactions set that to await and we're going to use db. transaction dot uh and then there's a

(1:13:31) method called find many okay and we can pass in there an object with a where Clause so with where we want to find where the user ID is equal to user ID all right and then let's see after that to in order to get the balance what we can do is we can take our transactions that we just fetched from the database and we can use reduce reduce is going to take in a function and that's going to take in let's say sum so the accumulator and then transaction and basically we just want to take the sum and we want to add the

(1:14:21) transaction do amount and then we just want to put the starting point which is zero so that should actually give us the balance and then we can return balance and if something goes wrong then we'll return an error and we'll just say uh what should we say database error I guess and then let's export this so export get user balance and yeah I think that that should do oh we want to export as default all right cool so let's go back into the balance component now and let's bring this in so up at the top we want to import get user

(1:15:23) balance and then how do we want to do this um so right here we should be able to say uh balance and we need to make this a sync because this is going to be a synchronous so await on get user balance all right and that should give us the balance and then come down here and let's get rid of the 500 and let's put in the balance all right undefined string is not assignable to type react node um what did I do wrong here oh this should be we're destructuring this okay and there we go 450 O2 now if we look at what we have as far

(1:16:29) as transactions if we go to our database so we got a paycheck for 500 and we got dinner for $49.98 so 500 minus 4998 is going to be 4502 now a couple things I want to do before we move on where we have the balance I'm going to use the nullish coalescing operator and put zero here so basically what this means is if whatever is on the left is null then it'll be the value on the right so if no balance it'll be zero and I'd also like to add commas if it's in the thousands so for instance if we say let's say paycheck two and we'll say it's for

(1:17:10) $800 and then we add that okay it gets added but I'd like to have commas in the thousands places so why don't we create a utility function for that and you could do this a million different ways but I'm going to go into the lib folder and create a file called utils.py okay and then what I'll do is return the whatever X is do2 string and then I want to do do replace and that takes in a regular expression uh actually we don't need quotes we need double forward slash now the regular expression I'm going to

(1:18:15) paste in I basically just use chat GPT for this just to get this expression to replace the commas where the thousands places so I'm going to paste that in we want it to be Global so we're going to add a g and then the second argument is going to be a comma whatever the character that you want to insert in the thousands places all right and then that should do it so I'm going to save that and then let's bring that into the component so import add commas and then down here we have the balance let's do add commas and

(1:18:53) we want to wrap the balance and the question marks in the zero okay so now if we check it out now we have a thand we have the comma where it's supposed to be now we're going to move on in a second but one thing I forgot to show you is if we go to our clerk dashboard you're going to see all your users okay so if I go to users I mean I only have one but you can see the user I mean if you were taking phone numbers you'd see that but we just have the email account we have the name we have the user image

(1:19:24) the user ID D so you'll be able to to check all your users from here and there's all types of other stuff you can do you can have you know organizations um and you can obviously change up the way that they log in if you want to add different uh like Facebook Twitter and GitHub and other ways of of validating you can do that as well uh but yeah I just wanted to show you that so now I want to have a spot where it shows the income and expenses cuz right now it shows the balance but I also want wanted to show how much income

(1:19:59) we have and how much expenses so we'll have another component and we're going to call that let's call it uh income what should we call what did I call it income expense. TSX and we'll say sfc whoops sfc and income expense and let's return uh div I'm going to give it a class of in c-- container and then inside that we'll have a div and an H4 in that div where it says income and then we'll have a paragraph with the class of money and the class of plus and let's say for now we'll just put $700 I'm just going to hard code it

(1:20:54) for now and then outside of that div we're going to have another div and that's where we'll have an H4 with expenses or expense and let's do a paragraph with money uh money and then a class of minus and for that we'll just say 200 again just hardcoding it for now and then let's put it into the homepage so I'm going to bring in import income expense and where we going to put this we're going to go right right below the balance okay so if we check it out that's what it looks like now we want to make this real so we want this to to come from our database so basically what

(1:21:42) we have to do is find all the positives add them together to be the income and then all the negatives and add those together that will be the expense so let's create another action and we'll call this get income expense. TS or expenses or no we can leave it singular get income expense now we can probably copy from the the uh get user balance action probably just copy all this because it's kind of similar doing a lot of the same stuff so we'll paste that in we're setting it use server we're getting the database getting the off let's change the name of the function to

(1:22:24) get get income expense and for the promise we're going to this is going to have income so income which will be a number and also expense which will be a number and then a possible error and then here I same thing we're getting the user ID we're checking for the user and then we want to get the transactions just like we did here but we're going to change some of this stuff up so let's get rid of that and what we want to do is first get the let's get the amounts so I'm going to say amounts and set that to transactions oops so we're getting

(1:23:10) transactions and then we'll use map and basically we're going to just get the just the amount because remember transaction is an object that has text amount ID user ID we just want the amount that's all we're dealing with so in the app let's add a function and we'll say for each transaction we just want to return the transaction.

(1:23:38) amount okay so that'll give us the amounts now after that let's get the income so we can take the amount or amounts and we can use yeah let's use filter and we just want want we basically we just want the positive right because the income is going to be positive num so in filter let's pass in a function and we'll say for each item we want a filter where item is greater than zero let me put this filter on a separate line so we'll do filter and then I'm going to use reduce and add an accumulator and the item um and then oops forgot my

(1:24:30) parenthesis so accumulator item and then what we can do is take that accumulator and just add the item because all we're doing here is adding all the the the incomes together and we want to start at zero all right so that'll give us the income then we want to do the expense so I could just I could just take this copy it down change this to expense and instead of we want to do filter but instead of item greater than zero we want to do item less than zero because we want the negatives and then same thing we're just going to add all those together so that'll give us the

(1:25:15) expenses now in the return instead of balance we want to get rid of that instead of balance we're going to have income and we're going to have expense um however expense I want to just wrap it in math. absolute or math.abs and then expense okay and then if there's an error we'll just say database error that's fine and then for the the export it's going to be get income expense so now let's go back to the component and let's bring in the acttion so we want to import income uh what is it is it get income expense and then we can get both

(1:26:05) the income and expense from that so let's destructure income expense from and we just want to make this a sync and then a weit and get income expense okay and then we're just going to come down here and replace the hardcoded numbers this one is going to be for income and this one is going to be for expense save that there we go we got 1300 income and 4998 expense and actually let's take a look here let me just reload this so yeah we got 800 and 500 which is going to be 1300 and then 4998 so that's working so the workflow is is very similar with all this stuff now finally

(1:27:08) we want to list the each transaction so let's create a new component and we're going to have a component called transaction list and transaction item for for this the individual item all right so let's uh let's create the list first so transaction list. TSX sfc transaction list and let's see we're going to uh how we going to do this let's just have a fragment and then we'll have an H3 and we'll say history underneath that we're going to have an unordered list with a class of list and basically I want to fetch the transactions Loop through them or map

(1:28:02) through them create a list and then have the individual transaction item component but for now we'll just output like the a paragraph with the with the text so I know we don't have the transactions yet but we will we'll create an action to get those so I'm just going to type out the code so transactions and let's say if there's transactions then we want to map oops read transactions.

(1:28:35) map and let's say for each transaction now we're going to have a type of transaction so I we'll create those in a minute too um and then let's have our arrow and then open up some parentheses and for now like I said we're just going to have a paragraph with the transaction.

(1:29:03) text all right so that's going to give us an error well it's not because it's not even in the the the app yet but let's bring it in so I'll go into page TSX and let's import transaction list and we're going to put this at the bottom so right under the form the add transactions so transaction list and now we're going to get an error now before we we go in and we create our action let's create the type for TR uh transactions so I'm going to create a folder in the root called types and let's create a file and call this transaction.

(1:29:42) TS okay and this is going to be pretty simple we're just going to export an interface and call this transaction and it's going to have have an ID string text string amount number um user ID string and created at date that's it so we can close that up and now let's bring that into where were we into our page. TSX because no not our page.

(1:30:25) TSX where were we um transaction list cuz we're using it right here so up at the top let's import um transaction and that's going to be from types okay so we get that now we need this we need our transactions so we're going to have a new action file called get transactions so new file get transactions. TSX so basically instead of API routes we're using using these server actions which I I I personally like and actually if you took my nextjs from scratch course the one I released a couple months ago we used API routes but I'm actually refactoring it to use

(1:31:09) actions instead and that should be released um probably a month or two so let's say use server I don't know why I keep saying user server and let's import the actually you know what let's just copy one of these we'll copy um the balance all right so let's change this to get transactions and let's see it's going to return let's change balance to transactions and that's going to be we can bring in the type as well that we just created so import transaction from types and this is going to be transaction but it's going to be an

(1:32:00) array so we want to put our brackets and then also a possible error and then we're going to get the user check for the user and then we're going to call find many so we're doing the same thing here we want to get only the user ID's transaction we don't need this balance get rid of that but in addition I want to order it I want to order it by date in descending so what we can do is after the wear we can say order by and say we want to order by created at and we want to do descending right and then we're just going to return instead of balance we're

(1:32:38) going to return transactions all right now make sure we change this too so get transactions now we can bring it into the transaction list let's go right here import get transactions and we'll go right here cons so isn't this better than use effects I I personally like like this a lot better than doing the the whole use effect thing uh let's see so I'm going to extract the transactions so transactions and error and let's set that to await get transactions which we need to make this asynchronous okay and then yeah I mean that should do it let's check for the error though so if we'll say if there's

(1:33:42) an error then let's return a paragraph and we'll give that a class of error and then just output the error okay so down here history we got paycheck to dinner paycheck so right now it's only showing a paragraph with the transaction text but obviously we want that to be its own component so let's create now um we can close up the get income expenses we can close up get transactions I think we have all of our components on the page we can close up page for now and income expense and then let's create a new component so in components we're

(1:34:30) going to have transaction item. TSX all right and a few things we're going to bring in so we're going to bring in the type of transaction so bring that in from types and then we're also going to wh we're also going to bring in the add commas and we're also going to bring in toast because we're we're going to have a delete button so I just wanted to show a toast that says it's deleted and then let's say sfc and transaction item it's going to have a prop of transaction that's going to get passed in so let's set that to the type of and then

(1:35:19) transaction colon and then transaction so we're using our type here all right and then in the return we're going to have list items so let's say lii now I want to have a class of minus if it's a negative if it's an expense and a class of plus if it's a a an income a positive number so what we can do is add a dynamic class name so set this to curly braces and we'll use a turn AR so we'll say if transaction do amount if that is less than zero okay so it's negative then we want a class of minus else we want a class of plus and then inside the LI let's have

(1:36:09) the transaction text okay and then underneath that we're going to have a span and in that span we'll put let's use our add commas so add commas and then inside that I'm just going to pass math ABS so that will get the absolute number of transaction.

(1:36:40) amount and I'm just going to we have more stuff to add to this but I want to just um just use this for now so we'll go back to the transaction list component and up at the top let's bring in transaction item and then we'll go down here and where we have this paragraph we're going to delete that and put in our transaction item and since it's a in a list we have to pass in a key and the key we're going to set that to transaction.

(1:37:15) ID and then pass in transaction and transaction save that and there we go so the reason you see the green for the the income and red for the expenses is because of this the plus and minus class all right cool now I'd also like to have the sign and if it's a positive it'll say plus if it's negative it'll have a minus so let's go into the the function but up at the very top above the return and let's create a variable called sign and we'll just set that to let's say transaction.

(1:38:03) amount if that's if that's less than Z then let's show a minus else let's show a plus and then down here right before let's see we're going to go right before the let's have the number sign here as well and then before that is where we'll put the sign so now you can see we have plus 800 and then minus 4998 now the last thing I want to do in here is the the delete so we'll have to have the delete functionality the delete event and then have uh an action to actually do the delete from the database so let's uh yeah let's add the button so under

(1:38:46) the span We'll add a button and we're just going to have an X in the the button and then let's add an onclick and we'll set that to a function and we're going to call a function called handle delete transaction and that's going to take in the transaction ID all right and then we're also going to add a class name so let's go outside of this curly brace here and then add a class name of delete - BTN and we have to create that handle delete transaction so let's go up above we'll go right under the sign variable

(1:39:33) and let's say handle handle delete transaction and we want to make this a sync and then this is going to take in transaction ID string all right and I I do want to have just a a confirmation so let's do const confirmed and we'll set that to window. confirm and let's say are you sure you want to delete this transaction all right and then we'll check that confirmed so if if not confirmed then we'll just return okay and then we're going to await on a function called delete transaction which we haven't created yet but we will and then that's going to take in the transaction

(1:40:47) ID and then after that we'll just do a toast. success and we'll say transaction deleted all right so obviously this delete transaction doesn't exist so we're going to create a new action so in the actions folder create a new file called delete transaction. TS and I'm just going to copy the user balance action whoops and we'll paste that in here and let's just change some things uh we do want to import the revalidate path because once we delete it's not going to update automatically unless we call this so let's say revalidate path from next

(1:41:39) cache and then let's change the function name so a what I do change the function name to delete transaction and that's going to take in transaction ID okay and then the promise that we return we'll just we'll just do a message string and get the ID check the ID and then in the try let's get rid of this stuff and what we're going to do is just await on DB do transaction.

(1:42:31) delete pass in an object and add our wear and we want to say where the ID is equal to the transaction ID and also where the user ID is equal to user ID all right then we want to just revalidate path just slash and then what we want to return we'll just return message yeah we'll say mage message and then transaction deleted okay that's it now let's go back into the item component and bring that in so import delete trans oh I didn't export it did I yeah we get the wrong thing exported so export that then we want to import that okay um now we're not doing anything with the message that's returned so we could do

(1:43:29) const we could do message and error and we can we could do if error then I guess we'll just uh we'll do a toast. error and pass in the error and then down here if there's not if there's not an error then we'll just say success but we can actually pass the message in that comes back from the server so that would be just message all right so let's see is this going to work event handlers cannot be passed to client component props um did I use CL oh yeah this has to be a client use client all right cool so if we come down here when I hover

(1:44:38) over it you see there's a delete button so I'm going to delete the let's just add another transaction I'll just say test and we'll say 300 okay so that gets added I don't even think we we tested that yet didn't we to see if if that showed up so it shows up test let's go ahead and click delete it's going to confirm let's click okay and we get transaction deleted and it goes away now it's just a couple small things that I wanted to do one if you look at the income 1300 there's no comma so we should use our add commas there and two

(1:45:16) I want to make sure that there's always two decimal places because there's a chance that we could add something and it just gives you gives us a bunch of decimals both with this component and the balance component so we can use the two fixed method and then pass in two to make sure that there's always two decimal places so let's take care of the the um let's do the balance first just make sure it's two decimal places so we'll go into the balance component and I'm going to add on to balance.

(1:45:46) two fixed and pass in two now two fixed will give you a string representation of the number with with the the decimal points and that's why we're getting this error because add commas takes in a number so what we want to do is just wrap this in number so open the parentheses there and then end ending parentheses here and that will get rid of that error and now it should always be uh two decimal places all right so the next one is this component which is the income expense component so this right here and we want to do this in both places where

(1:46:24) we have income and where we have expense now we need to bring in the add commas so we'll bring that in and then let's go to the income and we're going to do add commas okay so we're wrapping that now this income uh we're going to add two fixed to that and we're going to get the same issue because add commas needs a number so we want to also wrap this in number at our ending parenthesis okay and then we're going to do the same thing here so expense add add commas and then number and then addon to expense. to

(1:47:16) fixed fixed pass in two and add our ending parentheses all right let's save that and now you'll see that income now has the comma and we should always have our our two decimal places all right so that's it um one other thing I just wanted to test out was the the email and password sign up so I'm going to put in support at traversy media.

(1:47:54) com and then a password let's see we'll do continue and I'm going to have to verify my email so that's built in as well which is really cool I'm just going to pull this off screen and just verify my email real quick so it just gives me a verification code I'm going to grab that and paste that in here okay so now I'm logged in as my my new account and of course I don't see any transactions because I didn't add any with this account so I'll add let's say paycheck 500 and there we go that gets added now I wasn't going to do the deployment but I think we should just because it's it's easy enough with versel there is one

(1:48:42) thing we need to do since we're using Prisma so with versel it caches the dependencies of our project until one of those dependencies change and it does this to allow faster build and while this is a a good thing it does it can cause problems for the Prisma client so the solution is to just add a PO install script so if we go to package.

(1:49:08) json and in scripts we're just going to add to this and say po install and then we just want to run Prisma generate okay so we're going to run Prisma generate after in after deployment um and then that should work so let's close this stuff up and then we're going to go to versel now I already have this on on GitHub right here so just make sure that you deploy to either GitHub or bit bucket whatever service you're using and then I'm going to go to versell and click add new project and we're going to choose this project and then we do have some environment variables that we need to

(1:49:46) set so in the EnV let's grab the database URL value and here we're going to say database uncore URL paste that value in then we have the publishable key I'm going to grab that put that there and then the value put that there add and then we have the clerk secret key all right so we want the value grab that there we go so yeah I think everything looks good let's go ahead and click deploy so this is probably going to take a couple minutes so I'm just going to pause the video and come back when it's all done all right so looks like everything

(1:50:34) went through so let's continue to dashboard and then if we just click on this should take us to the live site there we go let's sign in now clerk is still in development mode so you can go to the dashboard and switch it to production but I just want to test out the login and make sure we should have all the TR same transactions if we use the same user because we're using the same database and that's up to you if you want to use the the same database in development as you do in production or if you want to have them separate but if I choose The Tech Guy

(1:51:11) info there we go so we have the same same exact transactions and everything is there let's try to add something new we say gas minus 50 okay so that gets added cool and we'll delete it awesome so now we have a deployed application so I hope you guys enjoyed this and you learned something from it and I will see you next time  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(58) Как Prisma стала ЛЮБИМОЙ ORM разработчиков? – YouTube

By RED Group

https://www.youtube.com/watch?v=tyCwTTcWcYE

Transcript:

(00:00) ребят Всем привет С вами Макс канал Red Group + и в этом шикарном видео мы с вами разберём и я вам покажу Почему Призма является номер один ормко на бэнде и почему её все так любят да И почему она в целом поменяла концепцию а взаимодействия с базами данных не забывайте реят поддерживать Данное видео лайком потому что ролик вышел невероятно годный и крутой и если мы набираем 500 лайков то я выпускаю следующий курс по Жун до и по Юта Вы давно этого просили а стейт менеджмента будущего скоро на канале кроме этого все вопросы можете задавать в комментариях Если они у вас

(00:33) вдруг возникнут после просмотра данного видео не забываем подписываться наш Telegram канал там очень много полезностей для разработчиков там очень много в целом голосовых сообщений очень много из жизни разработчика всё очень полезно и очень круто ребят ссылка в описании теперь у нас удобный alias Red Group очень короткий лаконичный QR код видите на экране буду рад вас там видеть Итак Всем приятного просмотра разберём сначала Что такое вообще в целом Омка Да и почему Призма является ормко Омка с ваше позволения прочитаю технология которая позволяет

(01:03) работать с базой данных используя объектовая взаимодействие между приложениями и базой данных автоматически преобразуя объекты в записи баз данных и наоборот or необходима для повышения продуктивности разработки улучшение читаемости кода и уменьшение количества ошибок при работе с данными если говорить кратко и моим языком раньше мы с вами помните в скле писали ка PHP времена PP были Да я сам был в этих временах сам писал запрос типа Select Where From и так далее Это мы всё помним но потом пришли Омки Они реально упростили Сначала я с ормко первый раз познакомился в лавеле в

(01:38) лавели есть своя ремка не знаю называтся как-то или нет названия У неё очень удобно обращаться к базе данных и вообще полный кайф а потом это всё и в жава скрипте появилось тоже И я сейчас пользуюсь призмой просто невероятно счастлив и кайфую То есть ты просто с помощью обычных объектов ты можешь с помощью обычной цепочки методов определённых ты можешь сформировать себе крутой запрос это очень и очень круто Итак Какие теперь у нас есть преимущество призмы перед другими орэм ками и почему всё-таки я в начале сказал что Призма - это номер один ом для базы данных значит смотрите

(02:08) Первое - это так называемая типизация либо типа безопасность э это когда Призма такое ни у кого нету то есть ну в плане у таких популярных ремок я такого не видел то есть когда Призма генерирует тебе автоматически все типы тайпскрипт на основе вашей схемы базы данных Что обеспечивает полную полную типизацию вашей базы данных и потом удобную работу с этими данными а также интуитивно интуитивно понятный API То есть вам очень легко делать запросы очень простецкий API очень понятный ээ причём он смысловой достаточно Да и я думаю что в этом будет разобраться крайне легко То есть я к призме привык

(02:44) прямо буквально за месяц когда на неё переходил также автоматическая миграция Да может автоматически генерировать и применять свои э применять миграции на основе изменений в вашей схеме тоже очень удобно а поддерживает огромное количество баз данных в отличие от других ремок она и постгрес поддерживает и mysql и SQL и насколько я помню вроде бы а монго de Бена тоже поддерживает если я ничего не путаю а также Призма созда позволяет создавать сложные запросы здесь немножечко Я небольшую оговорочка скажу потому что Да вроде бы сложные запросы но Как и любая рэмка она чем-то

(03:15) ограничена Да да я на своём опыте могу сказать что были конечно же такие задачи с которыми рэмка никакая не справилась бы да и мне приходилось ручками писать SQL запрос по-моему в этом ролике я не помню я в сценарии этого указал или нет мы этого обязательно коснётся Ну и конечно отличная документация документация призмы одна из самых лучших она новейшая Она крутая удобная Так что всем её рекомендую и на этом я думаю можно переходить к установке нашей призмы и запуске нашего проекта для начала ребята открываем терминал и пишем в нём команду для генерации нашего nestjs приложения значит я буду разбирать на при на примере nestjs Вы

(03:46) можете не обязательно sgs использовать можете обычную ноду призму позволяет и там и там главное чтобы в вашем проекте был typ SCP без этого конечно не некрасиво будет пишеш команду Nest а generate Nest Прошу прощения New я всё время путаю Nest New и называться будет просто Призма курс нажимаем Enter проводим установочный maner yarn устанавливается и мы сможем уже с вами после этого поставить призму нашу теперь Заходим в нашу папочку команда SD Prisma курс зашли и в этой папочке Давайте её откроем в V коде во-первых код

(04:19) Точка чтобы у вас работала такая команда Вы должны выполнить Command Shift P и нажать здесь и написать пав puf и выполнять данную команду Тогда эта команд код будет вам доступна О'кей поставили я git сразу выключу чтобы у нас не подсвечивать лишняя подсветка чтобы не был RM RF то git Готово А дальше ставим призму теперь пишем команду yarn Install точнее DD Прошу прощения призма и минус Def Да Def зависимость нажимаем Enter ждём установку обычно использую бан в проекте но в этом буду использовать yarn потому что не хочу смущать там новичков либо тех кто только входит в Кэн допустим да Поэтому я буду

(04:56) использовать самые примитивные технологии значит пишем Командо даже дальше npx Prisma а ИТ Да мы просто инициализирует приложение можно npx можно просто Призма если глобально она стоит У вас нажимаем Enter Всё у нас всё инициализировать вот вам подсказки пожалуйста Да во-первых у меня версия Как видите обновилась можно её без проблем Давайте Кстати обновим это уже вне ролика Итак что нам потребуется А значит нам нужно значить Database URL в NV файле сейчас Мы это сделаем потом нам нужно А если у Вас нету таблиц да то Ну

(05:27) короче если нет базы данных можно тоже создать а потом провайдер назначаем выполняем dbp если уже есть какая-то база данных Вы можете конвертировать из неё в призму схема Ну мы это делать не будем у нас будет пустая база данных и можно Prisma generate для генерации нашей базы данных об этом мы чуть дальше поговорим нажимаем C или Clear команда для очистки и всё примочка у нас поставилась теперь для начала мы перейдём в файлик а сма Призма Да и напишем нашу первую модель Ну а перед написанием модели расскажу про нашу образовательную

(05:56) платформу ребят Это очень важное предложение всего лишь действует 30 секунд ладно на самом деле реклама будет идти всего лишь 30 секунд предложение невероятно срочно смотрите А мы меняем полностью модель на нашем сайте подписки больше не будет то есть она закончится буквально в октябре месяце после этого будет продаваться всё только по штучно То есть каждый курс вы будете покупать поштучно курс мастер-класс всё что угодно поэтому у вас есть последняя возможность купить подписку на срок который вы себе хотите позволить там на месяц на полгода на год либо там на

(06:23) другие какие-то сроки Да покупайте подписку любой тариф который Вам вздумается и вы Несмотря на то что поменяется у нас допустим тариф на модели вообще её не будет подписки вы всё равно будете иметь всегда доступ ко всем интенсива которые там есть на момент действия вашей подписки то есть всё что будет выходить в будущем Если у вас есть подписка у вас будете будете иметь доступ к примеру скоро выходит у нас интенсив PEX S 155 и вы к нему также будете иметь доступ Если у вас есть подписка короче предложение крайне

(06:46) ограничено успевайте пока мы ещё не Обновили платформу ребят Вот уже совсем скоро подписка просто пропадёт ссылочка в описании там все плюшки там чатике полезные крутой коммьюнити поддержка мастер-классы интенсивы репозитории всё это есть там в нашей платформе ребят эта платформа разработана была лично мной она самая лучшая ссылочка в описании всех буду рад там видеть Итак Давайте напишем парочку моделей буквально две как описывается модель у нас в целом у меня будут иногда подсказки потому что я использую github Pilot для ускорения работы Но это никак

(07:17) не поменяет мои объяснение значит смотрите А как пишется Модели Модели очень удобная есть есть своеобразный язык в призме которые позволяет вам описывать ваши модели то есть модель - это является таблице в базе данных Давайте напишем module User и её раскроем будет у будет АШК будет непосредственно давайте мы кстати АШК сделаем значение мне больше так нравится да и ну хотя если мы идём Давай давай мы его оставим пусть будет потом на идт Ринго имя у нас идёт пароль если вдруг какой-то есть юзер у нас будет юзер без пароля и у нас также будут посты Давайте теперь

(07:56) сделаем связ юзера может быть много постов и также нужно привязать пост и сказать посту что у него может быть много юзеров давайте это сделаем Значит также ID что-то неправильно давай тут так string title ой Content Окей publish ну давайте я сделаю is pish типа опубликовано или нет default FS и также я укажу Map Map позволяет вам переименовать это поле в базе данных то есть здесь оно пишется ISP а в базе данных оно будет писаться как вот так вот да то есть это более правильно написание то же самое сделаю и переименую Также юзеры будут у меня users а пост у

(08:35) меня будет posts базе данных так авто ID Здесь всё готово Здесь тоже будет авто ID Окей то есть тем самым мы сейчас связали здесь мы указали что у нас будет аюр И связь будет на основе а ID Вот и референс будет Вот этот айдини то есть тем самым мы настроили связь между эти дву моде то есть пост имеет одного юзера а юзер имеет много постов сохраняем О'кей что нужно с делать дальше перейти и создать нашу базу данных я буду использовать pog Вы можете использовать свою базу данных для этого на сайте pog скачиваете инструментарий устанавливается на

(09:10) компьютер и открывать программку PG Admin дальше Заходим в Local нажимаем правой кнопкой на Database Create Database указ Base name давайте у меня это будет называться призма призма курс я скопирую сохраню всё всё успешно создалось можно закрывать закрываем И теперь мы это указываем в файли вот Database здесь собственно пишем логин ваший базы данных у меня это постгрес пароль пишем также постгрес ой прошу прощение 1 2 3 4 5 6 опять же все эти данные вы могли назначить либо если вы ставили через BR либо на винде как-то

(09:43) по-другому вы ставили через установщик какой-то Да вы всё равно должны знать сво логин и пароль от ваших базы данных Обычно они по умолчанию идут какие-то значения Вот и Local Host 5432 здесь менять не нужно название базы данных указываем здесь Призма курс и сма Public всё это можно удалить сохраняем закрываем теперь Мы открываем терминал и выполняем команду Призма Давайте npx npx Prisma db Push то есть мы как бы пушим изменения которые у нас есть в схеме в нашу базу данных чтобы мы в неё закачали что у нас

(10:17) будут две таблицы юзр и пост всё отлично зача всё успешно но Будьте аккуратны мы db Push используем только на де моде в режиме разработки Когда вы находитесь на продакшене всё работает только через миграции мы дальше в этом ролике поговорим о миграциях то есть имейте в виду что это работает немного иначе и э ну короче вам нужны миграция Да потому что миграция позволяет вам в любой момент откатиться А вот это пуш - это слишком жёсткая и строгая команда Вы сразу берёте пачкой перезаписывает все данные это очень негативно сказано на

(10:46) вашей базе данных и её целостности имейте это в виду поэтому используем миграцию обязательно миграция в Призма позволяет отслеживать вообще изменения А в коде Да как как ну в принципе я уже дал вам описание мы как раз-таки план в них перешли То есть вы сделали какие-то изменения поменяли какое-то поле Давайте этим займёмся как раз таки да и проведём миграцию нашу первую к примеру мы поменяли поле А давайте что-нибудь добавим какое-то поле у нас было контент как раз у нас нет поля дата создания поста created Add добавляем поле

(11:16) сохранили Теперь давайте сделаем миграцию то есть мы могли бы сейчас выполнить команду Push тем самым напрямую запушить Все изменения но это не безопасно с точки зрения целостности база данных поэтому выполняем команду npx Прима migrate Def А И через две чёрточки пишем name название нашей миграции у нас миграция можно init типа первая миграция дано у нас сразу же я покажу как нужно их называть вообще Обычно я это делаю следующим образом то есть к примеру если мы сейчас с вами говорим про добавление поля я пишу что мы Add created created Add to куда мы добавили to po po

(11:54) нажимаем Enter ждём какое-то время смотрите что у нас говорится говорится о том что у нас Drift detected Почему Потому что у нас до этого не было миграции и мы первую миграцию на катым А уже какие-то данные в ней есть и соотвественно оно говорит что у нас есть некоторые различия между между схемой Да и миграционными файлами потому что миграционных файлов у нас до этого ещё не было и видим что у нас добавлены две таблицы у нас изменено поле по изменено поле users и непосредственно мы и говорит вам о том Вот чем я люблю призму она вас всегда повесит если у вас идёт

(12:26) сброс базы данных вот сейчас она говорит о том что ты уверен что готов Потому что ты потеряешь абсолютно все данные ты потеряешь ты потеряешь все данные ты уверен что хочешь продолжить или нет Я говорю Да уверен потому что это мо первая миграция У меня пока что она пуста У меня нет никаких данных базе данных я могу себе это позволить поэтому делаю Yes Вот всё готово Ну вообще миграцию Конечно лучше накатывать когда у вас ещё даже когда вы не запушил ничего лучше накатить миграцию Вот но опять же я могу себе это позволить сделать потому что у

(12:52) меня короче у меня никаких данных нету И куда добавлять миграция миграция во-первых есть здесь вот може зайти в SQL и почитать Вот как раз таки то что вы описали в формате SQL выглядит следующим образом А кроме этого миграции есть Лок файл он пока что у нас пустой и в принципе в принципе на этом всё да каждая миграция имеет свой уникальный идентификатор и наше название которое мы прописали Кроме этого миграция также Вот будьте внимательны Да если вы будете вдруг удалять миграцию вам это поется в проекте Да и вы её отсюда удалили но при

(13:21) этом у вас она всё равно как бы осталась фактически и будет ругаться на эта Призма имейте в виду что миграцию также нужно удалить из самой таблицы у вас я м сейчас не смогу показать у вас в самой таблице в базе данных создаётся вот эти все таблицы которые мы указали Post users Да и будет также таблица migrations И там будут копиться все миграции ваши то есть вот там удаляете строку с этой миграции тогда всё отлично как будто её и не было Окей Чистим готово Итак после этой команды Мы работаем в деф моде на

(13:51) локалке непосредственно поэтому она автоматически примется вот мы выполним миграцию всё она применила в всё красиво но что здесь важно оговорить если как эти изменения теперь применить на продакшене вы запу проект в git выполнили CD к примеру и на продакшене вам нужно эту миграцию Развернуть как вы это будете делать для этого есть команда мы выполнять не будем но я покажу как она выглядит npx Prisma migrate и deploy вот эту команду вы выполняете уже непосредственно на продакшене и всё тем самым миграция красиво плавне

(14:21) применяется теперь перейдём к разбору крут операции с Призма клан значит Давайте закроем и интегрируем призму в наш ложение для этого перейдём в папочку sce правая кнопка New File значит Prisma TS готово и вставляем здесь код который официальной документации мы просто создаём класс Призма его расширяем на осно PR который нам Да библиотечка сама и имплементировать ум лишний файл отсюда будем прямо на ап контроллере Всё дело разбирать во-первых Нам нужно в модуль импортировать СМУ призму в провайдере запятую ставим и пишем Призма то есть мы

(15:09) будем использовать поэтому пишем Прима готово сохраняем закрываем теперь в Призма сервисе мы с вами Давайте чуть этот елин поменяю так добавляем свои две строчки всё чтобы нас это не беспокоило теперь переходим в а модуль Призма сервиса добавили отлично и теперь можно перейти в здесь с ним поработать вот у нас есть функция Get Hell давайте мы в ней попробуем получить всех наших юзеров Get users отдавать у нас будет конечно р из призмы берём про что говорил да типизация Вот видите у нас есть готовый полностью тип самого юзера очень удобно

(15:44) их будет много поэтому ставлю массив И тут Мы возвращаем выполняем здесь во-первых конструктор разворачиваем нам потребуется конструктор так и в конструкторе мы бем на пер возвращаем Ну либо Просто призму я люблю делать Да зачем серс нужен Прима ставим точка US и find ищем много юзеров всё смотрим какие-то есть конфликт Давайте уберём просто чтобы сэкономить ваше время готово также по такому же принципу мы можем создать что-нибудь давайте сделаем тут будем Давайте кани буде поть имя к примеру пусть будет имя и будем

(16:28) создавать users Create и что у нас требуется из обязательно поле name и email не забываем Да поэтому здесь не только имя но ещё также emil будем передавать мы сейчас ребят у нас был полный курс по nestjs на канале можете посмотреть мы сейчас nestjs разбирать не будем Разбираем чисто призму Как происходит запрос видите мы берём призму можно з is Prisma можно просто Призма Да эта мышь просто работает внутри класса поэтому используем zs можно просто Призма если у вас Кэн попроще в нём выбираем модель User Да и в нём выбираем find Man найти все найти много и так

(16:57) далее либо Create мед либо метод там find Unic есть есть update есть Delete всё что угодно значит указываете конкретную дату и в принципе всё И тем самым создали User ко по такому же принципу Я создам для обновления вам покажу Просто пример как это делается Вот таким образом да то есть мы получаем только дишни теперь мы его ищем ID у нас только это стринга не забываем Да не Number ищем и также его редактируем вот по Почему я говорю что человеческим языком написано как видите всё очень Тино Понятно update обновить Where найти

(17:26) где где что-то равно тому-то да да какую дату мы редактируем всё очень-очень понятно гляд И давайте для удаления сделаем Delete User принимаем дишни вызываем Prisma User Delete и указываем кого мы удаляем по шнику всё вот так работает наш крут теперь Давайте попробуем Ну давай сначала мы это выведем дело всё всё всё посмотрим А там уже дальше пойдём Так значит Get users Get users будет открываться по Так давай давай мы прямо здесь укажем users А почему бы нет вот будет открываться по главному руто вому пути

(17:56) здесь мы указываем Get users готово это мы убираем Давайте проверю наш запрос Я открываю терминал выполняю команду yarn ST Dev запускается запустило всё готово Отлично Теперь я буду проверять запрос в программки Rapid API она платная но у меня просто подписка на setup есть и поэтому я себе её поставил Ну так чисто потестить что-нибудь новенькое Вот это Абсолют неважно неважно Да многие пишут о том что мас У меня такого нету вообще неважно ребят можете инсомнию взять можете взять можете в самом коде Мы в

(18:30) других роликах это делали сейчас я хотел просто эту Ну просто для своего интереса скажем так пощупать не знать что это самое Там лучшая программа они все одинаковы плюс-минус Итак пишем Значит у нас по какому пути сервер наш открыт переходим в папочку MTS Local Host 3000 Ну пусть пусть так и будет не будем ничего менять будем оставлять default htp два США Local Host 3000 с нет У нас у нас нет никакого префикса поэтому сразу пишем SL users так готово нажимаем Enter и команда вроде бы выполнила мне не привы в этой программки работать немножко я к ней немножко не

(19:06) привык ещё но вроде бы что-то отработала то есть насколько вижу мы получаем J который абсолютно пустой Да данных нету потому что мы ничего не записали собственно это логично Окей но запрос прошёл успешно Хорошо давай теперь мы создадим нашего юзера а потом уже будем им заниматься значит для этого мы сделаем пост запрос на создание пост Create User будет получать имя email так то можно получать из B Да имейте в виду ники сейчас валидации мы делать не будем как я сказал потому что у нас нет на это времени B вот Body Body

(19:39) хорошо Так что нам ещё и сюда мы их передаём просто вроде бы всё так Аде давай сразу для удаления тоже вам покажу апдейт Я предпочитаю через делать мне так больше нравится Ну более правильно всё-таки тоже Аде и также только здесь Аде Я хочу вот через вот так взять поэтому здесь чучуть порек вот па ID через забираем также и тут мы тоже самое сделаем только для удаления то есть Delete Del ID Del US всё погнали готово Давай тестировать Теперь значит смотри переходим в нашу программку Дава о заем рядом плюсик готово вставляем значит здесь на будет уже пост запрос 3000 в принципе да Значит и данные

(20:39) данные нужно прокинуть у нас будет и сейчас с этим будем что-то что-то делать ребят Я пока не Как это работает через плюсик на да Ага и о странное Давайте раз чтом нас будет Допустим Максим мас Да и плюсик email будет допустим там @ test.ru отправляем запрос выполнился и данные нам пришли в ответе идеально проверяем юзеров теперь вот пожалуйста Всё у нас юзеры также пришли успешно вот он вот они юзеры пришли Конечно неудобно Здесь всё это дело показано визуал мне вообще не нравится в этой программе но

(21:22) мы в принципе т а вот мы можем вот так смотреть красивенько Всё Давайте постареть мне так больше Понятно Что происходит вообще в целом короче вот данные пришли Вот р нас записал базу данных Давайте мы его теперь удалим сейчас сразу все проверим запросы нажимаем плюсик так нужно Нижний Да плюсик походу нажимать Здесь тоже вставляем это уже будет Delete и так давайте посмотрим как у нас Del Ага У нас апдейт так идёт это неправильно Я не хочу так у нас вот так будет Ага И вот так будет ВС то есть мы ID нужно теперь нам узнать у нас ID вот здесь был вот он мы его копируем ко идём в Delete и вставляем

(22:08) сюда Всё мы должны удалить данного юзера нажимаем отправить Как видим пришёл ответ проверяем всё юзер успешно удалён всё работает Ну и апдейт собственно вы сами уже сдела домашнее задание вам будет Вот оно в принципе делается абсолютно Так же мы сейчас с вами проверили полностью нам неважно было у нас Мы проверили что крут операция работа успешно Окей теперь разберём с более такими сложными Зами фильтрация У нас есть интересный инструментарий который позволяет это делать К примеру Давай мы сейчас найдём опять же передавать я на ничего не буду здесь Но

(22:40) мы типа так формально передадим Мы же всё-таки призму изучаем Да как будто мы сюда ко передали значение Мы хотим найти юзеров у которых имя будет Алиса пусть будет реально пусть будет Алиса имя Алиса Давайте Попро найти для это пишем можно сделать без проблем во-первых у нас может поменяться здесь значит Алиса может поменяться с маленькой с большой буквой это нужно тоже учитывать в данный момент и и допустим это может быть там Алиса Там какая-то фамилия у небудь там н илинь допустим да к примеру То есть у не может какая-то фамилия быть а мы а мы передали чисто Алису и мы в итоге не сможем найти

(23:19) как сделать правильный поиск то есть считайте мы сейчас делаем просто обычный поиск нашем приложении для этого мы Давайте да то есть вот эта штука должна включаться наше имя но сейчас она чувствительно к регистру То есть если мы с большой буквой передали А здесь базе данных с маленькой то ничего не найдётся ставим запятую и указываем здесь что нам нужно мод мод мы указываем здесь insens insens то чтобы он был нечувствительный А давайте будем проверять У нас сейчас по-моему пустая Да краски база Давай сюда перейдём добавим нашу Алису С

(23:57) маленькой букво добавим отправляем готово проверяем Алиса готова Вот она при этом а ну мы уже в принципе просто да сделали то есть у нас смотрите мы с большой буквы ищем при этом находит давай проверим вообще что это в целом работает пишем Алис о поиск Как видим ничего не нашлось потому что Алис оди не существует А вот а так теперь не важно с маленькой с большой буквой Оно всегда будет находить это очень удобно это реально очень удобно Также вы легко можете здесь ить всякие параметры То есть вы можете написать in

(24:28) к примеру и если у вас Ну как раз таки те самые наши посты Да к примеру то есть мы с вами можем incl и если какие-то посты у нас были бы то мы мы бы их развернули Давай сейчас я попробую на примере это подсказать давай мы когда создаём мы создадим посты значит Create один пост создадим пока что временно и у него будет H и там вроде итен какой-то контент видишь в этом gpt очень сильно не gpt помогает очень сильно всё готово какой-то пол Давайте ещё раз мы создадим пост Алиса с большой букво Эс Давайте Ладно давайте на Макса

(25:14) перейдём Макс Пусть будет так у нас здесь уникальный тест о готово запустили Теперь мы если получим значит наших Давайте Сначала уберём это поле чтобы показать как без него работает так сохраняем и тут мне ну мы ищем Макса ищем Вот Макс прил Да но пока что как видите Мы у него не видим никаких постов вообще они есть или нет мы пока не знаем поэтому мы здесь добавляем значит in обновляемся Как видим теперь посты включены мы видим полноценно весь пост нашего Макса более того мы можем писать не не получать все поля получать конкретно каждое поле

(25:57) допустим приме тоже можно сделать Ну я сделаю Select и получу только title делаем запрос Как видим получаем только title очень очень удобно а это касаемо продвинутого инструментария то есть мы изучили incl Select это вы всё посмотрите что важно ребят Вот эти курсы которые сейчас выходят в новом формате вот этот следующий курс тоже потом посмотрим по вашему фидбек что будет получаться они очень короткие они такие длинные как раньше ни по три ни по 5 часов и так далее Почему Потому что я для себя понял что вам же нужно лишь

(26:26) познакомиться с технологией дальше уже у вас индивидуальный будет кейс и типа зачем вам сейчас рассказывать всю базу которую вы в итоге не Будете учить и вы просто забудете без практики поэтому я даю вам некоторую такую базу вас как-то мотивирует на эту технологию А вы её у себя уже интегрируйте скачиваете наш проект домашнее задание после этого ролика без проблем и ну и там как бы да всё это дело дело производите и там можно накапливать каким-то опытом конкретным Теперь давайте разберём транзакции значит смотрите Для чего

(26:56) нужны в целом транзакции И почему они важны давайте я делал отдельный метод вот так резко мы к ним переключились рассказываю Ребят давайте синхрон функцию примат готово Что такое транзакции ребят рассказываю значит они обеспечивают целостность данных при множественных операциях в принципе этим всё сказано Давайте русском языке обя новички не понимат зачем нам нуж транзакции влом такие типа ну мы ж можем просто вот убрать транзакцию Да и сделать раз Create два Create в чём проблема а если вдруг у вас много людей на сайте в данный момент находится Да и в момент когда вы делаете какой-то

(27:39) запрос другой человек делает такой же запрос может произойти сбой баз сбой работы баз данных и нарушится некоторая целостность потому что два одинаковых запроса и они могут могут конфликтовать скажем так да поэтому тем самым что мы обернули всё в транзакцию мы чётко знаем что мы должны сделать два запроса они как бы превращаются в один запрос да И если в одном будет это ошибка а второе не сделается то есть оно делается просто в совокупности вместе и поэтому транзакция обеспечивает жёсткую безопасность и то есть это очень сильно актуально Когда у вас завязаны друг на друге к примеру

(28:09) А смотрите у нас пост очень сильно завязан на вот этой вот Алисе То есть пока мы Алису не создадим нам пост не к чему привязывать будет то есть и поэтому целостность нарушится то есть не может быть такого что мы с вами условно Ну если бы мы без транзакции делали мы бы Вдруг у нас произошла здесь ошибка а здесь всё мы не можем нас будет ошибка потому что нас мы мы юзера то толком не создали поэтому объединяем их в один запрос это очень важно И в итоге получаем вишь две функции на выходе а О'кей это касаемо транзакции Теперь давайте мы

(28:41) попробуем для сложных операций описать некоторый SQL запрос как вообще SQL запросы пишутся в призме смотрите я напишу какую-то допустим асинхронную давайте без асинхронной нам хватит Просто значит SQL request такой запрос базовый Да как он будет делаться пишем This Prisma тодор quy Row и теперь через обратную кавычку или как она называется Я уже забыл мы можем спокойно описать запрос вот уже подсказки некоторые есть давай сохраним И будем его писать я в своей практике так скажу что наверное один проект у меня приходится порядка Ну на такой небольшой проект порядка Ну где-то пять-шесть запросов SQ

(29:20) У меня есть почему потому что они очень сложные всякие джоны используются То есть это невозможно или всякие там числительные операция на уровне SQL запроса в сделать с помощью ремки это невозможно Поэтому приходится сэлем иногда описать значит Какой пример я хочу вам написать к примеру Мы хотим найти всех Ну кто не знает SQL Да кто вдруг новенький SQL раньше так все писали в принципе да вот просто брали вот этот кусок и его писали базу данных и в принципе когда мы пишем омку а она в конце В итоге преобразовывает всё в SQ

(29:50) запрос для базы данных это важно понимать что давайте я буду его расшифровывать Select Мы выбираем всё звёздочка значает Всё вы можете указать конкретное поле можете все поля выбираем всё от юзера Ну не просто от юзера А у которого у которого допустим Ну пусть будет email равен Ну либо можно сюда вставить какую-то переменную Давайте типа я вот так вставлю Чтобы вы потом в будущем могли поменять допустим тест собачка тест Вот то есть мы с вами в итоге дава так как бы мы это сделали если Взяли бы обычную мку смотрите Return This Prisma точка find сейчас User find Unic

(30:37) потому что уникальное поле и мы бы с вами сделали где email Test собака test.ru так и закрываем то есть вот вот такое различие да то есть либо мы сделали бы вот такую штуку либо вот такую штуку Да но опять же повторюсь что вот В чём минус Да она не оптимизирована у неё нет типизации и Также важно понимать что м ну типа это этот код он будет не масштабируемый то есть в О он ужасен для масштабируемости Вот Но иногда это нужно делать потому что он имеет больше возможностей вы напряму обращаетесь к базе данных всё это вы вы исходников

(31:14) скачаете можете потом посмотреть теперь Вишенка на торте Ребят я вам расскажу о новой функции кото о которой меня все мне все говорили в комментариях под курсом по njs о том что Макс ты упустил один момент я ребят Не выпустил Просто я его записывал заранее до выхода буквально несколько дней до выхода обновления ПО призме Призма наконец-таки презентовала пишем Призма Давайте Призма separate сма Files они наконец сделали обновление их Их Очень долго просили которая 3 июня это было которая позволяет теперь разбивать вашу схему на

(31:50) очень много разных маленьких схем это очень удобно Я уже успел это пощупать Давайте опустимся ниже и будем это интегрировать всё смотрите у нас слева мы увидим нашу схему Давайте её откроем А наша схема вот она и у нас Казалось бы у нас очень простая схема но вы наверное уже могли догадаться что если бы в нашем проекте было бы условно там ну как в обычном проекте не знаю у меня там получалось там до 300 строк кода примерно прикиньте 300 строк кода это максимально нечитаемо Вам скролить приходится искать это неудобно и все долго и и мечтали просто о том что

(32:23) Призма это интегрировано поэтому Давайте этим займёмся как это всё работает мы во-первых Давайте сначала будем дробить на разные файлы А ну юзера вынесем в юзер то есть пишем нажим New File User Prisma важно называть коротко в документацию всё это дело почитаете вставляем готово и также с постом копируем пост точка Призма вставляем готово Так хорошо тут тоже сохраняем пока что ошибки Это нормально кроме мы мы забыли сма сделать Давай тоже сделаем New skim и в неё перенесём всё пост сма и User всё переносим сюда и в самой схеме мы должны указать поле preview futures

(33:12) это обязательно то есть не здесь Прошу прощения а вот здесь вот здесь сохраняем всё После этого у нас всё стабильно работает Давайте мы выполним Призма Прима dey к примеру да а у нас сечас Не сработает я неправильно писал Давайте напишем при всё Как видите у нас Нана одна миграция и она не требует применения то есть всё у нас стабильно работает но теперь мы разнесли всё по файлам сепарировать модно молодёжное слово да разделили и у нас вместо вот того чтобы была каша в одном файле теперь можно разделять на красивые понятные файлы это

(33:49) очень удобно так что Пользуйтесь обязательно переходим к заключению сначала перед заключением хотел сказать что всё вы с скать ссылка описание по нашей общей подписке любой тариф покупайте и вы сможете скачать а любой Ну как бы любой исходник там 100 120 репозиториев уже по-моему или 115 и в том числе эта репозитория тоже будет там то есть вы сам проект сможете легко скачать по ссылке описании Итак в этом курсе Мы рассмотрели основные концепции Давайте ещё раз вспомню что мы рассмотрели А значит создавать Как

(34:17) создавать управлять схемами выполнили миграцию круто операции все прошли выполнили сложные задачи применяли фильтрацию работали с транзакциями Для обеспечения Для обеспечения целостности ваших запросов также мы делали SQL запросы также мы разбивали схему на более мелкие компоненты и в принципе вот так наверное наверное И всё да что в итоге Призма - это очень мощный интуитивно понятный инструмент который Вам реально поможет вашей разработки Так что я крайне Рекомендую вам попробовать использовать его в проекте а продолжайте

(34:48) практиковаться и и исследовать в принципе возможности призмы она очень большая и там очень много есть крутых штук которые мы сегодня не разобрали Ну это и не нужно разбирать потому что вы в проекте у себя это окунёво это всё с полноценной практикой это будет гораздо лучше чем я тут буду вам теорию заливать которая в итоге никуда не ляжет на этом всё ребят Не забывайте подписать на Telegram ставить лайки комментарии писать и конечно же не забывайте покупать нашу подписку пока есть такая возможность Всем спасибо ребят Всем пока