Introduction To Tiered Applications

Video Transcript

Video 1 – Getting Started With Back-End Development

Hi, my colleague Abel Sanchez and I are excited to present this course on back-end development. It's the final of three courses, bringing together what we call the full stack. After this course, you should be able to develop apps and make decisions about what tooling and what packages you need to develop your app with. We'll be covering the full three tiers, starting at the front-end, going through the web servers and the middleware to the back-end databases. We'll have two major flagship projects. One is called Bad Bank. And that'll be a full three-tier application of banking where we have to deposit and withdraw from our bank account. Now, any application today is probably going to be deployed to the Cloud.

So, we're covering topics like Docker containers. We're covering DevOps and the DevOps pipelines. We'll be looking at how these pipelines are architected. So, today we talk about loosely coupled pipelines so that each one can be updated independently. And this is how DevOps manages to deploy, in some cases, 20 times a day. What's the tooling needed to do that? We'll be covering that from your editor VS Code through GitHub and GitHub actions, through continuous integration and development to deployment in the Cloud. The tools that are needed to, for example, test the code, test the front-end, test the middleware, testing the backend.

We'll be implementing our second major flagship project that involves a shopping cart. And we'd be placing items into that shopping cart, taking items out. And finally, when we check out, we'll be billing a credit card using Stripe. There are many, many packages that we're going to be using. I think we counted roughly 20 of them. And you're going to understand the advantages and disadvantages of each one. So, you'll be able to make decisions about what you need when you develop your app and when you deploy it to the Cloud. As I say, we're really excited by this course, and we think you will be too.

Video 2 – What Is The Back End

What is the back end? It sounds a little, a little mysterious, a little bit like when you go shopping and people talk about the back room where they have extra things, and things come together for the front-end of the store. Well, when it comes to computation, you can think of the client-server relationship in much the same way. You have a front-end, this is what customer see. This is what is your front-end in the browser. And then you have a back end where multiple things come together. You can think of it here very much in the terms that you think about it in the store. There's the pretty part, the part that is arranged for customers to come and be able to identify their products and purchase things.

Then there's that back room where things come together to be able to create that, that shopping in the front-end that experience. But technically, what are we talking about? I'll talk here about four broad categories of what the server, the back end can be. This can be much more sophisticated, much richer, but we can have some broad grouping of functionality under these four categories. And that is server. You can have one or multiple. That is, performing the function of responding to that front-end, providing additional services. Then you can have the second category, which would be data stores.

Here you might have different types of databases, different types of persistent mechanisms. Then comes the programming language and the operating system environment. Once again, you're dealing with multiple pieces, and there are a multitude of these as well. And the last component is the API. We're living in a world of services. You can think of them as building blocks that are coming together. And you reach out to all of these using APIs. So, there you have it. That is what the backend is. Those four broad categories, server, database, programming languages, operating systems, and APIs.

Video 3 – Introduction To The Three-Tiered Application

Creating a three-tier application, or for that matter, an n-tier application can be quite challenging. There's a tremendous number of design decisions, there's a great deal of architectural considerations. And because of it, we're going to start off here with a simplified case. We're going to look at three little tiers. We'll look at a simplified client that is we will not look at the full spectrum of possibilities, but we will look at one that will allow us to have a client. We will then move to the server-side, and although there can be great complexity there, we will look at a subset of that functionality. We will then look at a small, tiny little database and do the same. And although this is a simplified scenario, three little tiers, you will see that it will allow you to look through this spectrum of possibilities, look at this number of tiers and get a first step in creating that entire system, that entire solution that considers all of those tiers. And you can take that same knowledge and expand it onto n-tiers.

Video 4 – Server

In this module, we're going to be creating a client, a server, and the associated storage, the so-called three-tier application. As you can see here in this diagram, our client is going to be the browser, our server is going to be an HTTP server. And then we're going to have a data store. Now, these applications can be challenging because there are a lot of moving parts. And because of it, we're going to bill one piece at a time and give you a chance to see them all separately, and then bring them together as we progress. So, as you can see here we're going to start off first by creating a very small little server. That server is going to use the Node platform, it's going to use in an npm package, which is the registry for Node packages called Express, which is very popular. And then for the client to that server, we're going to write, we will use the Chrome browser.

So, let's go ahead and get started. We'll start off by creating first the Node application. So, let's go ahead and move to the console. As you can see here, I have a terminal window open and I am on the desktop. I'm going to go ahead and create a directory, and I'm going to call it 'littleserver'. And now, I'm going to go ahead and move into 'littleserver'. Inside of it, I'm going to go ahead and initialize this Node application. We do that by entering 'npm init', and it'll ask us a series of questions about this application. The first one being, do we want to go with the default? Which is simply the directory name. We will take that the version, the description, we will say, 'small server'. Then the entry point which is the default will be '(index.js)'. No 'test command:', no 'git', no 'keywords:'. We'll enter my e-mail as 'author:'

Then enter the 'license:' of ' "MIT" ', and confirm that that is the information that we want. I'm going to go ahead and list my files. And as you can see there, that initialization created a file called 'package.json'. If we take a look at it, you can see there that the data that we just entered is now part of this file and that is in the JSON format. Now, once we do this, we're going to go ahead and move to the registry and take a look at the express package that we mentioned that gives us the functionality that we're looking for in HTTP server. And here in the browser, in the address for the registry is npmjs.com. And we can go hand and look for express, we should note that there are a great deal of packages, well over a million, so it's a good place to explore when you're looking for functionality.

In this case, we're looking here for express, as mentioned, you can see here that it's a very popular package with over 14 million downloads in the last week. You can also see there that you get some starter code you can link to the GitHub repository. You can read through the documentation from the rest of the page. But in our case, we simply want to get here the information on how to install it. And you can see there that the instruction or the command it's 'npm i express', and 'i' is short for install. So, let's go ahead and move back to the console, to the terminal. Now, we're going to go ahead and install that package. And you can see there that being done. Now, if you take a look at our 'package.json', you'd see there that we have now a new dependency. And this is done so that if somebody wants to recreate the work that you've done, they can recreate the dependencies that you have by running npm install and using this package.json.

Now, at this point, we have everything we need and we're going to go ahead and move to the editor and write our small little server. As you can see here, I've opened up my editor and I'm going to go ahead and drag and drop LITTLESERVER. And we're going to go ahead and take a look at the files. And you can see there that the files that we have, are package.json, a second one that is supporting the dependencies. That one is auto-generated, you don't need to touch it. And then we have a folder called node_modules. That is where all of the dependencies that Express have, get added. This is not something you need to manage by hand but just simply for completeness, I'm going to go ahead and expand. And you can see there the dependent, that Express has a great deal of functionality and dependencies when it comes to other packages.

So, at this point, we're going to go ahead and write our littleserver. As noted, when we were creating the metadata, and I think I wrote, I pressed in the wrong link. No, that's the right one, but it is not in the right place. I'm going to go exit from there. That is, it was being created inside of node_modules. and I want it to be at the root. I'm going to go ahead and enter the name here, index.js, which is the convention for the default file in a Node.js application. And I'm going to go ahead here and create an instance of express. So, I'm going to go ahead here and enter 'express', 'require', which is the keyword to bring in that package. And then I'm going to go ahead and create an app. And that is where we create an instance of 'express( );'. As you can see there. And once we do that, we have the ability to create routes. That is paths that are defining an API that you can then access through a client.

So, I'm going to go ahead here and enter a get for the default path, which would be the root. And then this is going to take a callback. And this callback has two parameters, which are usually called, request and response. And they're typically entered as 'req' and 'res'. Now, this is the function that is going to be called back. And in this case, all I'm going to do is that I'm going to respond and send the following. And all we're going to do is that we're going to say '('Hello World!');' as it's usually done on a Hello World application. And once we did that, we've defined a path, we've created an instance of express, now we need to create a listen. And so, we can do that by entering 'app.listen'. I'm going to go ahead and enter the port that I want to listen to here.

And I'm going to enter a callback simply to give myself a message of where this server is running. And I'm going to enter here '('Running on port 3000!');'. I'm going to go ahead and save this file. And now, I'm going to go ahead and go back to the console. And now, if we list our files, we can see there our index.js. I'm going to go ahead and clear the console here. And I'm going to go ahead and enter 'node index.js'. And as you can see there we're 'Running on port 3000!'. So, let's move to the browser. And let's go ahead and enter our path, which will be, in this case, localhost:3000. And as you can see there, we get our Hello World!. So, to recap, we created a folder, we initialized the Node application, then we installed Express. We wrote a little bit of code to create an HTTP server, and then we accessed that server through a client, which was the browser.

Video 5 – Data Store

In our previous lesson, we created a littleserver. In this case, we're going to create a little database. Just as before, we're going to use a package from the npm registry. The package that we're going to use is called LowDB. Now, this is a very minimal database. However, it illustrates the importance of being able to persist information, and with the goal of our lesson of not being overwhelmed, we're going to bring in the small package that simply creates a JSON file that we can use to persist our data. So, let's go ahead and navigate to the npm registry. And as you can see here, we get some information about this package called LowDB. You can see there the weekly downloads. You can see how we can do a 'post' that is right to this file that we're going to be using. You can see further down here how do we get started. We require the package.

We require the 'FileSync'. We give a name to the file that's going to be persisted into our file system, that is going to hold the JSON data that we're going to be writing. And you can see there how we use this file, and how we can write to the file. And later on in the documentation, you can see how to lead, how to filter, and some of the other features that you will want to do when you're working with data. Ultimately, the crud, as it's often called, the create, read, update, and delete that old data applications tend to have. As you can see here, I've opened a terminal window on my desktop, and I'm going to go ahead and 'clone' a repository. Go ahead and 'clear' the screen, and I'm going to go ahead and move into the 'littletiers' folder. And as you can see there we have some starter code. Now, let's go ahead and move to the editor. And I'm going to go ahead and drag and drop that folder.

And as you can see there, we get the same listing of files. Now, this is going to be the repository that we're going to be working throughout the rest of the lessons in this module. And I'm going to start off here simply with the little db part, the small database. And as you can see there, I've already entered some of the boilerplate to use this package '('lowdb');'. And I have a number of other tasks. They are listed. Now, I'm going to go ahead and do a couple of the initial ones, and then I will leave the rest up to you. Now, to get started, let's go ahead and take a look at the documentation. And as you can see there, we have comments here for initializing the 'data store', adding a 'post', and the other ones, I will leave for you.

So, let's go ahead and move to npm and take a look at how do we get started and how do we add a post. And as you can see, there are the first steps. So, I'm going to go ahead and copy this, paste it into my editor. And I only need, well, actually I'm just going to remove this one here. And this is going to be my 'defaults', so in it, I'm not going to use 'user:'. So, I will remove that, and I will put the rest all in one line. So, as you can see, the first one sets a array of post. And then on the second one, I'm going to go ahead and '.push' into '('posts')', the following entry. An 'id:' of '1,', a 'title:' of ' 'lowdb is awesome' '. And I'm going to go ahead and add one more property, I will add the property of 'published:'. This can be true or false. So, I'm going to go ahead and say 'true' for this one. And that's all I'm going to enter, to get started.

So, let's go ahead and save this. Let's move back to the console. Now, I'm going to go ahead and run just 'simple.js', as you can see there. And when I run it, you can see there that it says 'MODULE_NOT_FOUND', and that is because I have not installed yet, lowdb. And I can do it two ways. The very first way is I can just do 'npm install lowdb'. That's the name. Or I can trust the person that wrote the repository, which is, in this case, it's me, and do npm install. And the reason that would work is because if we take a look at package.json, you'll see there that I have ' "lowdb" ' as a dependency. Now, in this case, I'm simply going to go ahead and enter 'lowdb'. And you can see there that it was installed. Now, if I were to do 'npm install', it would do the same thing, but add express as well. So, I'm simply going to do it twice, so you can see there that both ways can be used.

So, now that we've added both packages, let's go ahead and run our code once again. And this will be simple.js, and as you can see there that ran. And if we take a look at the file system, you'll see there now that there is a new file called 'db.json'. And if we go here to our file system, we can see that that file is here, db.json. And you can see there that that data was persisted. What that means is that if I were to run something else, I could read that information directly from the file system because it has been persisted into a permanent storage, a permanent data store. Now, let's go ahead and make some more entries here. And now, I will be careful here because I don't want to repeat my ids. And it will simply make three more entries. There are ways to avoid doing this by hand, but we're simply doing a very simple example.

So, I will do it by hand, and I will enter here titles, And it doesn't matter what I write. So, I will say ' 'random' ' here. This makes some of these 'false' just to get some variability here. And let's not forget to remove the first one since that one was one we already wrote. And nothing will happen by the way, if you enter it is just you will get two entries with id. And it's sort of defeats the purpose of having an id if you are repeating it. So, let's go ahead and run our code one more time. Now if we take a look at db.json, you can see there that we now have three more entries, the ones that we specified. Now if we wanted to read that back, I'm going to go ahead and remove everything that I entered here. And I'm going to go ahead and enter 'db.get('posts').value();'.

And I got that from reading the documentation. And I'm going to go ahead and save it. Let's go ahead and rerun that code. Oh, and I forgot to write to the console. So, let's wrap this in a 'console.log'. Let's go ahead, and run it again. And as you can see there, as expected, we can read our data back. So, this is simply emphasizing the fact that the data is being persisted across sessions. And I will leave the rest of the tasks on the file for you to do. So, count your post, find post ids, and you can see there some of the additional task. You might want to be able to delete one, say given a certain id, you can play there a little bit with the functionality, and get comfortable with this package as we will use it in following exercises or in following lessons.

Video 6 – Server Plus Data Store

In previous lessons, we did an HTTP server on its own, a little server. Then we made a small data store, a little db on its own. Now, we're going to be bringing both of those together and providing an integrated service that's going to be able to read the data and write to the data store. When it comes to the packages, we're going to be bringing LowDB, which is our small data store, and Express, which is our small server, in this case. As mentioned before, both of these are packages that I found through the npm registry, the registry for Node platform packages. And we will be accessing and verifying that functionality that we do on the server, and that writing and reading from the database by accessing through the client. In this case, the client will be the browser.

We're not yet going to do UI. We will simply be working through the address bar, and hitting those routes on the server.

So, let's go ahead and move to the console first. And as you can see here, I am inside of 'littletiers'. This is the same project we have been working with. If I list my files, you can see there the files we wrote before, 'simple.js', which was the one we use to write to the small little datastore. Now, in this case, we're going to go ahead and verify that we have the packages that we need. These are packages we used before, both Express and LowDB. And if we took a look at our 'package.json', you would see there that we have those ' "dependencies" ' both to ' "express" ', which is going to give us the HTTP functionality we need, and also ' "lowdb" ', which we're using for a data store. So, if we wanted to install both of them, we can simply enter 'npm install', and you would get the packages.

Now, it doesn't matter if you've done it once or if you've done it 10 times, it will work every time. You can also install any one package as well if you wanted to. Do it one at a time, as I will do here, 'npm install lowdb'. Then I can go back to simply installing both of them. I just wanted to show you there that it is pretty robust, and it doesn't matter which order you do it in. It's convenient to have it in the 'package.json' because then anybody else can replicate the dependencies that you added. Now, if we look back at our file system, you'll see there that I have a file called 'http_server.js'. And in this file, we will be bringing both of those components or those two sets of functionality, meaning the functionality of being able to have a server, and the functionality of being able to persist data into the same file, which will be a server.

And we'll have the capability to read from that data store and write to that data store. So, let's go ahead and move to the editor and take a look at that file. And as you can see here, I have my project and I'm going to go ahead and open up http_server.js. And as you can see there, I have placeholders for some of what will be your exercises and some other parts that we will do now. So, as you can see here I have 'add http server' in the first lines. Then I have a number of routes of paths, addresses that will be matched when you type that into the browser. In this case, the very first one will be ' '/data' ', and it will return the data that is in LowDB. And you can see there are a few other ones. At the bottom of it as well, I have a line here to start the server. These are things we did before, but we're now doing them together.

So, we're going to start off here by creating or adding the code to be able to run our Express server. And I'm going to go ahead here and start with 'express'. And this is the variable name that I'm going to use. Then we're going to go ahead and 'require' that package, which we have already installed. And then we're going to go ahead and create an instance of it. We will call that instance 'app'. This is a convention. We will go ahead and enter, 'express();', and create an instance. As you can see there, I have used 'var' for the other variables, or you can change those to 'const' if you like, in this case, it's a small example. So, we can go ahead and use 'var'. Now, once we've done that, then we need to start our server, and that is down here at the bottom. And I'm going to go ahead and enter here 'app.listen'. And I'm going to run the server on port '3000'.

And I will give myself a friendly message reminding me which port this is running on. So, I will enter this at the console, 'console.log', and I will write '('Running on port 3000!')'. Now, at this point, it would work, and it would listen for request. However, we are not doing anything on any of the routes, and so we want to return some data to be able to see what's taking place. So, let's go ahead and do the very first one, which is the one for ' '/data' '. And here we're going to respond. So, this is 'res.send()'. We're going to send it to the client. The result of reading are small 'db'. So, we're going to go ahead and look for the property of '('posts')'. This is something we've done before, and we're simply going to read all of them. And that's all we're going to do. So, I'm going to go ahead and save this. And I'm going to go ahead and move to the console and run the server.

Now, I'm going to go ahead and enter 'node http_server.js'. And as you can see there, we have a port running. We have a server 'Running on port 3000!'. So, now we're going to go ahead and enter that path, which is just localhost at port 3000 slash data, and see if we can read that data back. Now, let's go ahead and enter that into our browser. I'm going to go ahead and enter localhost:3000, and then data Let's go ahead and see if that work. And as you can see there, we get the data that we have entered before the "id"s, the "title", the "published", and so on. So, that's pretty good. We've gone ahead and brought together these two pieces, the persistent, the datastore, and the serving, and the paths that allow us to interact with a client. And we have easily shown here the data that is now right and residing within that JSON file, within that data store.

Now, in addition to that, let's go ahead and do a write, this was a read. So, let's go ahead and move back to the editor. And we're going to write the second, the second path, as you can see here, the second route. And note that it has a specific structure. It's 'localhost:3000/posts' then 'ping/1' and '/false'. Now, 'ping/1' and '/false' corresponds to 'title/:id/' and ':published' '. And we're going to use the packaging that the framework does for us, to be able to extract those. Now, to be more explicit, let me go ahead and show you that in a diagram. And as you can see here, I have color-coded those components. I have title, id, and published. And you can see below, an example there. The path is localhost:3000/post. And then what follows after that are parameters of data. In this case, the title, the id, and publish which are Hello, 9, and false.

So, let's go ahead and collect that data in a, in an object that we're going to write. Let's go ahead and make it 'post'. And inside of it, we're going to have three parameters. Well, let's start off with 'id'. And we have access to the parameters that have been posted using 'req.params', as you can see there in the auto-complete. And then the name of your params. And in this case, it would be 'id'. The next one is going to be the same except for 'title'. Then the last one is going to be the 'published' property. Let's go ahead and enter 'published,' there. And simply to be JSON, fully a 100% JSON compliant, I'm going to go ahead and put those variables in quotations. And now, we have a 'post', an object for a post. And so, what we're going to do now is that we're going to go ahead and get a handle on our '('posts')'.

Then we're going to go ahead and 'push()' a new object onto the posts. And so, in this case, it's going to be simply '(post)'. And then the last part of this is 'write();'. Now, this is functionality that LowDB defines. And if you want to see more about it, you can read out the documentation on npm. Now, we're also going to go ahead and write to the console, so we can see it. Go ahead and enter 'console.log'. And then we will read as we did before, everything that is in the database to see if we have gone ahead and written as we think we have to the data store. And then the last thing that we're going to do is we're going to respond to the client and see, and send that information back as well. So, we can confirm it from the client as well. Let's go ahead and respond here, and we will enter the same. And that's all we need. So, let's go ahead and save this.

I'm going to go ahead and take my example here, which is already in the right structure. And I'm going to go ahead and move to the browser. Let's go ahead and paste that address into the browser. And when I hit return, nothing happens. And the reason why nothing happens is because we have not stopped the server, and reloaded the new functionality, the new code that we have written. The code that is running, that is based on the older code that we had. So, let's go ahead and move to our console. Let's go ahead and stop the server. And let's go ahead and start it again. Now, we'll go ahead and enter here the new address. And I think that went through because the browser was waiting to post. And as you can see here, we have now an "id" of "1", which I know is redundant, but this is just an exercise that we're doing, and the "title" of "ping", "published", and "false".

So, let's go ahead and enter another one, and let's call this one, "wow". An "id" of "6", just to do something different, and then "true". And as you can see there, we have written to the database. Now, if we go back to the file system, to our console, you can see there that that's been echoed to the console as well. If we took a look at our 'db.json', you'd see there that we have all of that data that we have written as well. A final note on parameters and routing. If you want to read more, you can go look at the documentation for Express.

And to conclude, let me go ahead and make some comments, and let's go back to our editor for that. And so, as you can see, that was a fairly easy thing, low effort. We had already done the functionality or done an exercise with the data store in one with a little HTTP server. And here, we just brought both of them together. Now, when it comes to data applications, we often talk about crud, create, read, update, and delete. And so, you, we did the create, we did the read. Take a look in how to do the update and how to do that leap. I'm giving you there some placeholders for filtering and some other functionality. You can go ahead and add your own routes, as I mentioned if you wanted to do update and delete. And now, it's your turn. Go ahead and give it a try.

Video 7 – The Three Tiers

In our previous lessons, we wrote a small HTTP server, then we wrote a small data store. We then brought both of them together and accessed the routes on the server through the browser client. However, we did not write a user interface. In this lesson, we will do so. So, we will continue to use what we have on the backend, that is the packages from npm running on top of Node, Express, and LowDB, the functionality that we have built. But in addition to that, we will now have a front-end. And that front-end will be an HTML page, and that HTML page will be leveraging a couple of libraries. The very first one is Bootstrap, the biggest collection of styles or the biggest framework for styles on the web. And the second one will be Superagent. Superagent we will use to make calls on behind the scenes on the HTML page to the server. And we will be using it because doing low-level calls is a pain and Superagent can handle that for us.

The other one is mentioned as Bootstrap, and this is simply styles and you can read more about them if you like. Now, we will be serving static files. That is the user interface files from a "public" directory under Express. Now, it's worth noting that you do not need to do this with Express. You could have a separate server that simply serves static files. But in our case for convenience, we will be doing it within Express. And you can see there the line, the instruction that needs to be added for us to be able to serve files from a "public" directory. And the "public" directory is one that is inside of our code. And we can see it in a moment when we move to the editor. So, let's do so now. As you can see here, we have our listing of files inside of little tiers, the project we have been working on. And if I expand public, you will see there that I already have an index.html.

Now, I'm going to go ahead and add that instruction first to http_server. And you can see here that I have a placeholder for it. Now, let's go ahead and paste that line. And as you can see there, it's using 'app', which is 'express', and then it is denoting a directory called '('public')' that we have here in our folder. And it is writing the name there as you can see. So, let's go ahead and open up index, I'm going to go ahead and wrap the text. And as you can see there, I have a link to those two files. The very first one is the styles from 'bootstrap', as you can see here. The second one is that library. So, you can see for 'superagent'. I then have here a big button that we will see in a moment once we take a look at that page, and then below that, I have the data or not the data, but the code for making that call to the server.

And as you can see here, this part refers to 'superagent'. And we will take a look at that in a moment. But first, let's go ahead and run our server, save our file first, run our server and confirm that we can serve that file. So, let's go ahead and run the server. And you can see there it's listening. Let's move to the browser. And in the browser, we're going to go ahead and enter localhost:3000, and then we're going to go ahead and enter index.html. And as you can see there we get a nice big button and the styling of the button comes from Bootstrap. And if a took a look at the code behind here, I'm going to make my font a little smaller, you could see the code that we had in the editor. So now, we're going to go ahead and write that code. And as you can see here, the code is going to be invoked by the 'onclick' on the button, and it's going to call 'data()'. This is our 'data()' function.

And now, let's go ahead and move back to the editor. And so, here we have the same code within the editor. And you can see there that I'm getting a handle on an element called ' "status" '. We will use that shortly. The other one is setting the path. This is the path for ' '/data'; ' that will simply return everything that we have in the data store. You can see here that 'superagent' is setting the address that we are going to target within the server. And the rest of it is the code to handle the response. And the response, as usual, is handled with a callback. And this callback is going to have two parameters an error, in case we have one, then the response, we're going to handle both of those cases. 'if' we have an '(err)', we're going to go ahead and write to the 'console' the '(err);'. Then we are going to handle the second case, in case there is no error. We will simply write to the 'console' the '(res);'. So now, let's go ahead and go back to our browser.

Reload that page. Confirm that we have the code that we have added and we do. And so now I'm going to go ahead here and open up the developer tools. I'm going to go ahead and make this bigger. So, we can see the message. Let me go ahead and make my font larger as well. Let's go ahead and click on Show All Data. And as you can see there we get a response. Let's go ahead and expand it. And I happen to know that the response comes back in the body. And if you look inside of the body, you can see there the data from the server. Now, at this point, you'd want to take that structured data and write it in a pretty way to the UI. Here, I'm simply going to demonstrate that we can in fact do that by using the status element. And all I'm going to do is simply write to the inner HTML of the 'status'. And so, that is 'innerHTML'. And I'm going to use 'JSON.stringify', which will simply create a string from our JSON object.

And so, in this case, that is going to be '(res.body);' because that's where that data is. Now, let's go ahead and go back to the browser, reload the page to get the new code. And when I click on Show All Data, you can see there that we are now injecting into the UI that new data. So, to recap, we're doing quite a bit in this last exercise, or in this last lesson, we're leveraging the HTTP server that we wrote, the data store that we wrote. And now, we have built a user interface for it. So, we're going across all three tiers and we're leveraging a number of tools. We're leveraging Express on the server-side, LowDB on the server-side, part of Node.js platform. We're also using a browser client and we're using a couple of tools there. One for styling, the other one for making calls. So, make sure you understand the flow of these different tiers, what runs on each place. And for an exercise, go ahead and address one of the other routes and make sure you can do updates, and update your UI as well to reflect what you have on your data store.

Video 8 – Receiving Data From The User Interface

You will often want to collect information from users, and a very convenient way to do it is to use forms. We will extend our exercise that we have been working on to be able to take input from a user to create a new post. So, let's go ahead and move to the code. And we will use our index.html as a template. Let's go ahead and save that file under post.html, a form to create new posts. And we're going to go ahead and remove the current code. And we're going to go ahead and add a number of 'input' elements. And each of these is going to have an 'id="" ' and a 'placeholder="" '. Now, I will copy it over three times. And then the very first one is going to have an 'id' of ' "id" '. The second one is going to be the ' "title" '. The third one is going to be ' "published" '. These are the fields that we have within our data structure. So, ' "id" ', ' "title" ', and ' "published" '.

We will then add a '< button >'. And this '< button >' is going to be of 'type="text" '. And I think I made a mistake there. Yes, the 'type=' of button is ' "button" '. And I'm going to go ahead and add a 'class='. And this one comes from Bootstrap. This is a class to make a nice blue button. Then 'onclick='. I'm going to go ahead and call a function called ' "post()" '. And now, the text that will be visible in the interface will be '>Summit<'. Next, I'm going to go ahead and write that 'post()' 'function'. So, I'm overriding the function that we had before for data. And I'm going to go ahead and get a handle on those elements. So, I will enter three of them. The very first one will be '('id');', the second one will be '('title')'. Third one will be '('published')'. And I want to catch those in variables that have the same name. So, this will be 'title' then 'published'. And in this case, we don't just want to get a handle on the element, but we actually want the value.

So, I need to enter here 'value;', '.value;' for each one of them. And I'm now going to take advantage of string composition in ES6, which allows me to add placeholders for those variables, as you can see there, and construct the string. So, that's all we're going to do. We're going to keep it pretty simple. I'm going to go ahead and remove this line here so we can take a look at it at the UI. And let's go ahead and load that into the browser. So now, we're going to go ahead and enter localhost:3000/ and the new page is post.html. And as you can see there, we get it all in one line because I have not added any line returns. And so, I'm going to go ahead and enter here 11, just to enter something the title is going to be, say, posting from the web. And published is going to be false.

Let's go ahead and open up our developer tools so we can see that message. And let's go ahead and take a look. Yes, we got a response. And as you can see there, we entered, we have a mistake. So, let's go ahead and take a look at our code. And yes, as expected, there's something wrong with the way we get '('id')'. That is because we did not get 'id'. And that should be enough. Let's go ahead and go back to the browser. Let's reload the page. Enter a new id, a new title. Let's say, hello world, published, in this case, true. Submit. Let's go ahead and take a look at our 'body'. And as you can see there now we have a, we have a posting as expected.

Now, take a look at something that we haven't yet made a comment on. The first one is that if you look at the ids, 1 through 4 are as expected, they are numbers, but 4 through 7 are actually strings. So, this is a good example of how data can get dirty. There is a mistake in the logic there. And I will leave that to you to fix as an exercise. As well as round out, as well as round out the rest of the functionality for crud. We did create, we did read within that do update or delete. You can create a form to do that. You can create some listing of the data that you have. You can play around with the functionality that you have here on this application. There's a good amount there to do. You have the logic on the server, you have the richness that potentially you can have in the UI. So now, it's your turn. Have some fun.

Video 9 – Jest Hello World Test

Let's take a look at testing and using Jest to do that. So, the first thing we need to do is install Jest on our local machine. So, we run 'npm install jest'. So, let's go and do that. Okay, and let's take a look now at 'package.json' file. And we'll see we've got ' "jest" ' here, installed. But we now need to alter this script that we need to change it to ' "test" ', and then ' "jest" '. Usually, it won't have ' "jest" ' there. So, you need to change that. Now, that'll give us the ability to run 'npm test'. And you will see it's testing my file, and it's passed. So, let's take a look now at the structure of the files that are needed to test. So, Jest will look for any files that have .test.js. So, here we've got hello.js, and that's what we want to test. And so, we create a file hello.test.js. So, 'hello.js', just contains a function 'hello', that returns ' "Hello World" ', and we write 'console.log'.

Now, we need to export that. Here in the test, we import it. We use 'hello = require("./hello");'. We don't need to put the js on this 'hello' here, even though the file is called 'hello.js'. And we're importing 'hello'. And to actually get at the function, where down below here we run 'hello.hello()'. And it should be equal to '("Hello World");'. If it is, we'll pass the test. Now, in our index file. We'll also have index.html, and that contains other code but uses hello.js. Let's just take a quick look at that. So, here's our index file. We import ' "./hello.js" ' here, and we use it to run 'hello()' here. And we can run now the tests as we showed here, we can run 'npm test'. And that'll run the test for us. And we're going to use this later, obviously, in our automated testing. But that's an example of using Jest.

Video 10 – Jest SuperTest

So, let's take a look now at testing a Node Express web server. We're going to use Jest and Supertest. And a web server is normally listening on a port. And we don't want it to actually be listening when we do our testing, we'd like to be able to run multiple tests in parallel, for example. And we don't want to be spinning up, say, lots of web servers, that are listing on the same port. So, to get around that, we're going to have to modify our code a little. So, let's take a look at that. So, here's our main app server. And normally, it will be listening on port 3000. Now, to test it, we need to remove that and insert this 'module.exports = app;'. And then in the test, we're going to put in that we 'require' the '/.serverApp'. Okay? So, let's look at the code. So, here we are in the code. And I want you to focus on serverApp.js and the serverApp.test.

So, here's our serverApp. It's basically the code that we really want to run. So, I've stripped that out of this. So, I just want to test the main part of this, which is the roots. So, I've stripped those out and put them in here. And as I say, instead of listening, on this, we do an export and now for our server test we 'require("supertest");'. So, we're using this package that will send requests to our endpoints. So, here, for example, we're doing a test which is sending a request. And notice it sends 'request', and '(app)' is the argument. But we're doing a '.get' on that '("/");'. And here we're doing a '.get' on '("/contacts")'. And then we do a '.post' on '("/contact")', and we test the responses. So, take a look at these. It shows you how we can set up these tests. Let me just scroll them so you can see. And now, we can go and just run those tests in. So, here we are. And so, let's take a quick look at 'package.json'.

So, inside 'package.json', you notice we've got ' "supertest": ', and we've got ' "jest": '. And there's quite a few others that we've got, and here we're testing; we can just do 'jest' this was to detect if we had any open handles. So, for example, if we were listening on ports, running the actual app, this would detect it for us, but basically, we're doing 'test' on 'jest', so let's run that 'npm test'. And we'll see that we passed all the tests. So, one of the things that we can do is use '--watch' with 'jest' running which is quite nice. Now, what this will do is if we make any changes to our files. So, let me make a change, for example, to one of these files. So, I'll save this, and you'll notice it ran on the tests automatically. So, 'jest' with '--watch' will automatically run the tests, which is very neat indeed. So, these are a little bit more complicated to test web servers than test ordinary application code. But Jest does a really good job, and especially when it's paired with Supertest.