BACKEND

What exactly is a Full-Stack Web Developer? Well, it's the combination of knowing about how to work with the frontend, so the client-facing side and also how to work with the backend, which is the side that's invisible to the user, but contains all the logic that makes a web application work. And this is the part that we're going to be learning about in the coming lessons. Now, I love this cartoon that describes exactly what is the difference between the frontend and the backend. Notice that here is our user and he's looking at this beautiful mermaid, and everything the user can see is a part of the frontend. It's the font, it's the colors, it's the typography, it's the buttons, the forms, everything that the user can see, or hear, or interact with, is a part of the front end of the website. Now, what about this dragon at the back? Well, because the backend is the part the user will never see and never interact with, sometimes it can get a little bit monster like, but it doesn't have to. And we're going to learn all the things that's going to make our frontend as great as our backend. But essentially the backend is the part the user never gets to interact with, and never gets to touch or see. So what exactly is the backend? Don't worry, it's not anything rude.

It consists of three main components. It's the Server, which is basically just a computer, the Application, which is all the logic that enables the web app to function, and finally, a Database where you can store and retrieve user data. Now let's look at each of these in a little bit more detail. As I mentioned before,

a Server is just a computer, and a server for a live website should be on 24/7 so that it can listen for incoming requests from browsers from all over the world. Traditionally, servers are made and optimized for this particular purpose, but at the end of the day, any computer that is connected to the network can act as a server. And in fact, when we're going to be building our websites locally, we're going to be using something called localhost, which means we're going to be using our own computer as the server in order to serve up the websites that we create. And even though it's not on 24/7, or maybe you keep your computer on 24/7, it doesn't matter because we only need it to be on and connected when we're testing it.

The second aspect of the backend is the Application, and the application is basically Logic that runs on that computer, the server. And this logic determines how you want to respond to the requests from the browser. Depending on what that request is, let's say a user clicked on a button and that request came through the server to the application, well, the application knows that that particular button should then return a particular HTML page. So then it sends that HTML page back to the browser and the user gets a look at a new web page. Sometimes an application can also respond with different things other than HTML. So it could send back Data or it could send back just a Status Code. And you've probably all been on websites where you've seen

the status code that is a "404", and this is a code that the application on a server sent back because it's trying to tell the browser that that particular request you sent is invalid. And what that means is the application does not know how to respond to that particular request, the browser made. So maybe you tried to go to a weird URL that doesn't exist, or you tried to click on something that's broken. But either way, when the application doesn't know what to do with the request the browser is sending over, it will send back an invalid, which is a 404 status code and then usually your website will just show the user, "Oops, I think you did something wrong."

The final aspect of the backend is the Database. And this is not a requirement of web applications, but usually as your web app gets more complex and larger, then it's going to need to start to store user data and that is what databases are for. So when we store the data onto a hard drive, it's kind of like saving your word document, right? If suddenly the power went out and you didn't save your work, then you will probably lose it. But if you hit Save, even if the power goes out the next time you log on, you'll still be able to see your work. And this happens exactly the same way on a server. If that computer lost power and you didn't have any sort of database, then all of that data will get completely wiped unless we have a way to persist the information. So you can view the database as a form of permanent storage for your web application. Now, I keep talking about this idea of a Web Application, and the reason for that is because so far what we've been building, websites that only have a front end, they are essentially web pages, they are bits of HTML that gets requested from the browser to our server, and the server then returns that HTML file which gets rendered on the browser. So it could include the HTML, the CSS and the JavaScript. And this is why we've been able to use GitHub pages to serve these static web pages because all we needed to do was to store these files somewhere and when the user requests it by going to a particular URL, then the server, which in our case was GitHub pages, then returns these files back to the browser and gets rendered for the user to see. Nothing very complex there. But when we're talking about a Web Application or a Web App, then in this case it should be able to do something beyond just allowing users to view web pages. Let's think about a really, really simple type of web application functionality, and that is simply just allowing users to log in. Let's say the user goes to our website, our web app, and enters their email and their password because they're trying to log in to our website. Now, these two pieces of information are passed to our server in a Request. Because our server is on and is listening for requests, it's able to find the application that knows how to handle the login. So maybe it'll be like a login.js file. In this file there will be some logic that knows how to log a user in and that logic is usually pretty simple. We could look inside our database, look for a particular email and get hold of the password for that email, and then we can compare that password that's in the database for the user's email against the password that they sent us from the frontend. And if those two pieces of data match, then we can send back via the server, a Response in the form of, say, the Home page. Right? So home.html maybe. And if for some reason it didn't match, then maybe our response would be to send them back to the login page and get them to try again with an error code. If you think about this whole flow of our web application, things are a little bit more complex, and the reason why it is more complex is because we now have a web application with logic and a front and a backend. The frontend is everything the user sees, the backend is all of the things that the Server, the Application and the Database does in order for our website to have more complex functionality. If that's all a little bit abstract and difficult to imagine, then you can think of a restaurant as a really good analogy for this process of frontend and backend. If we divide up a restaurant into a frontend, which is the actual restaurant that the user or the client gets to see, and then we have all of the rest over here as the backend, then we can start thinking about what happens when we order at a restaurant. Firstly, you enter the restaurant. Here you are a little stick figure, looking beautiful as always, and you say to the waiter, "Hey, I want something." So you take a look at the menu, which essentially is all of the functionality that this particular web app has. And you can also think of this as just all the different interactions on the website, buttons, or forms, and you decide to click on a particular button, let's say, right? So you click on this button on a website or in the case of our restaurant, you tell the waiter your order from the menu, and then what the waiter is going to do is he's going to traverse from the frontend to the backend. And this is what happens when your browser sends that request to your server. The server is effectively the same as the kitchen in the restaurant. It's open 24/7, well, certain restaurants are anyways, but in this case it's listening for the requests that's coming from our waiter, from the front-end, from the restaurant, and in the server there are applications, which is basically our chefs, and what our chefs are able to do is they have lots of recipes. They know exactly how to respond to different requests from the client. So let's say in this case, the client wanted some pancakes. Well, the application, our chef, knows how to make a pancake. It has the pancake recipe. It checks the recipe it sees, "Oh, we need some data from our database." If we imagine the pantry as the database of the restaurant, then what the chef or the application is going to do is going to request those ingredients or those pieces of data to process it or work with it somehow and then at the end, it's going to send back a response to the front-end. So in this case, we got some pancakes that were made in the kitchen and sent into the front of house, which is the restaurant, and in our server that could be some HTML, CSS and JavaScript files that got created or retrieved and sent over to the frontend. Now, if you wanted more food or a different item, then you can again, ask the waiter to make another request to your server. And this cycle of response and request between the frontend and the backend is essentially how most web apps work. So the next time when you're thinking about how a Full-Stack Web Application works, you can think about the client who's a person at a restaurant who is ordering food, and that request gets relayed to the kitchen, which is the server-side. And in the server there's an application which is similar to the kitchen that prepares the order and working with the data in the pantry of the kitchen, and once all of that logic is processed, the food, the actual thing that the user should see is then sent back to the client side where the response is the food on the actual plate.