**Overview of the Functionality of the Code**

**What this Web Application Does:**

Before going into the nitty-gritty technical details of the code, we first have to get a general overview of what this application is. Basically, all this code does is to create a web browser-based sign up form. You input your email address for the username and then you input the password you want to use. This email address and password combination is then stored in a database. Then on a login page, you can sign in using that same email address and password combination once it is retrieved from the database.

            So, in summary this application creates a sign-up page that stores an email and password that you can then use to sign in to the web page.

**General Overview of the Technical Aspects:**

The functionality of this web application seems simple enough. All of us who have access to the internet have had to sign up and log in to use a website. However, this simple login functionality is considered to be a full-stack application.

**What’s a Full Stack Application?**

            A full stack application is defined as having three parts: The front end and back end.

The front-end part that consists of the web page or User Interface. It’s basically the part of the web page that interacts with the user. This would include how the website looks and is displayed on the webpage and how the user inputs data to get what he/she wants. Using an automobile as an analogy, the front end would be like the steering wheel, brake pedal, acceleration pedal, shifter and all the other buttons/switches that the driver presses to drive the car, open windows/doors, use the radio, or whatever.  The cosmetic looks and design of the car like the car color would also be considered front-end.

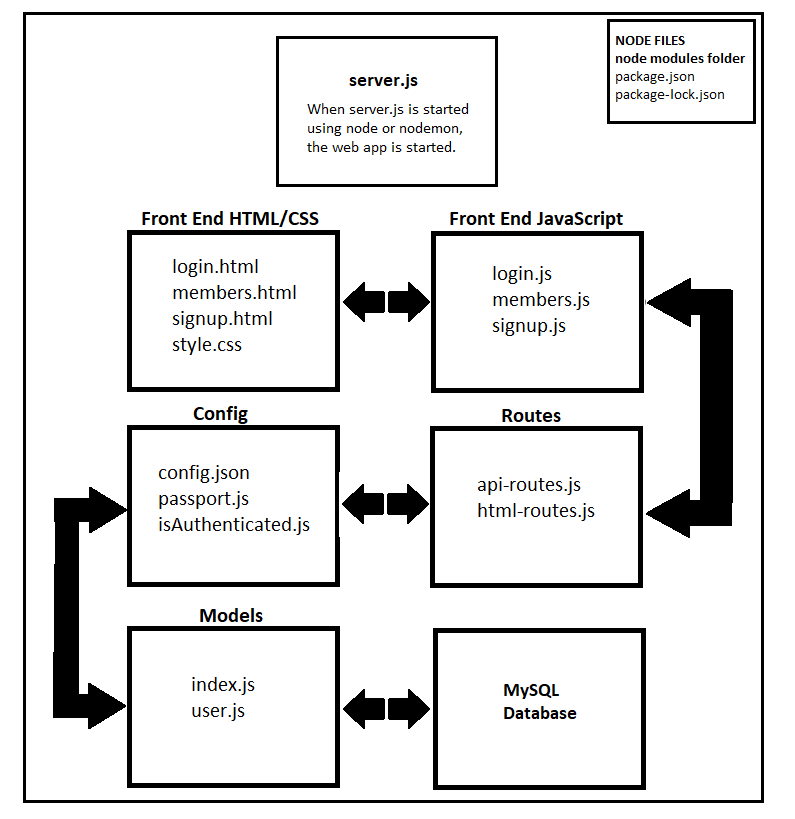
The back-end of the web application consists of how to store the data that the user inputs and how to retrieve that data from a separate remote computer called a server. It’s all the stuff that goes on behind the scenes of the program that the user doesn’t see… Unless something breaks. Using the car analogy, the back-end would be tuning the engine, and transmission in a car to get them to work together smoothly and efficiently without blowing up.

**How Full-Stack Applies to our Specific Application:**

The front-end part of this application is sign in and login page where the user inputs information and then logs in.

The back portion of the application consists of how to store this user created email and password combinations into a SQL database and then retrieve them when needed.  The data will be stored in a remote server, so part of back end programming is not just storing, retrieving and editing data, but also how to securely establish a connection between the user’s computer and a remote server that might be thousands of miles away or maybe hundreds of thousands of miles away for a computer in outer space. How do we create a secure link between the user and server? How do we store and retrieve data between the user and server?

The rest of this tutorial will focus on the functions of each of the files of this application.

**Overview of the File Structure of the Login Page:** The code is divided into the following structure:

As you can see in the picture, we initiate the application by using Node to initiate the file named “server.js”. The application is then structured into the following sections:

**Front End HTML/CSS**:

            The Front-End HTML/CSS is the text and graphics that the user will see on his/her web browser. It is the interface used to view data retrieved from the server, input requests to the server, and to send data to be stored in the server.

Next, we will discuss the functions of the following files that are part of the Front-End HTML section:

·         **Login.html:** This is the html file used by the user to login to the web page using an already created email address and password.

·         **Members.html:** This is the html file used to welcome the user once he/she logs on.

·         **Signup.html:** This is the file used by the user to input a new email address and password combination.

**Front End JavaScript:**

            The Front-End JavaScript files are used to provide functionality to different elements on the webpage that the user will interact with. When the user interacts with a specific element on the web page, the Front-End JavaScript will initiate a route to or from the Server data or to a different web page. Depending on the route, data will either be created and stored in the server or retrieved from the server to display to the user. Also front end JavaScript is used to add cool  functionality to the web page that might not be possible with just CSS and HTML.

Next, we will discuss the functions of the following files that are part of the Front-End JS section:

·         **Login.js:** creates functions that are called when the user interacts with different elements of the login web page.

·         **Members.js:** creates functions that are called when the user interacts with different elements of the members web page.

·         **Signup.js:** creates functions that are called when the user interacts with different elements of the signup web page.

**Routes:**

            The Routes files create pathways from the front-end web page to different items stored on the server. These items could be other web pages, a video, or data stored in a database. There will be a different route created for each. There are different routes for the Server to communicate back to the user(client-side) and for the client to communicate back to the server. This is the equivalent of building roads between different cities or physical locations on Earth.

Next, we will discuss the functions of the following files that are part of the Routes section:

·         **Api-Routes.js:** This file creates different routes between the front-end web pages to the MySQL database. Different parts of the web page have different routes to the database.

·         **Html-Routes.js:** This file creates routes between different web pages stored in the server.

**Config:**

            The Config files create functions that involve communication between the Client and Server. Examples of the types of tasks that the Config files deal with are what to do when a user inputs an incorrect password, what to do if the user requests data from a database that doesn’t exist, the connection information needed to login to a database, what routes the user is allowed to visit, etc.. It’s like establishing the ground rules on how a relationship between two people are conducted.

Next, we will discuss the functions of the following files that are part of the Config section:

·         **Passport.js:** uses the node module named Passport to create various conditions when the user wants to to create and use a login that is linked to a database.

·         **Config.json:** a JSON file that contains the login information to connect to a particular MySQL database.

·         **isAuthenticated.js:** a function that makes sure the user is logged in before he/she is able to access a particular route to data stored on the server.

**Models:**

            The files contained in the Models section of the program create functions that interact directly with a database… In this case it will be the creation of JavaScript functions that can interact with a MySQL database.  The code in the Models section of the program are intended to be a middle man between two different languages. In this case, it will be between JavaScript and MySQL. And between MySQL and different filetypes stored in the application. This code is created so that a JavaScript developer doesn’t have to know MySQL or to interact directly with a MySQL database. There are already prebuilt functions that perform the tasks. These model functions can then be reused across different applications without developers having to know more than one language.

            In this case, the middleman that we will use is a module called Sequelize. Sequelize contains many prebuilt JavaScript functions that perform different tasks in an SQL database. If one knows how to use Sequelize, there will be no need to learn how to query an SQL database directly.

Next, we will discuss the functions of the following files that are part of the Models section:

·         **index.js:** creates functions that allow a connection between the SQL database and external file types. These functions make it so that a JavaScript developer doesn’t have to interact directly with file types and can just call a premade JS function.

·         **user.js:** creates functions that allow a connection between  the SQL database and the inputs on the front end webpage. This file will encrypt the password that the user inputs for a particular email and stores that encrypted vale into the SQL database.

**MySQL Database:**

This is the actual database that stores data and can be edited and queried from. A Database is a table or series of tables that is able to store a large amount of information in an organized manner that is useful for a user. This database is stored in the Server, and the Client can use the web browser to interact with this database.

            In the case of this web application, the database here is used to store usernames and their corresponding passwords.

Next, we will discuss the functions of the following files that are part of the MySQL Database section:

·         **passport\_demo:** This is the name of the database that will store usernames and passwords.

**Node Files**

This section of the code contains all the node modules and their dependencies that are required to run this application. There is a package.json file that lists all the dependencies that node will download automatically when a command is called. The package-lock.json file makes sure that when Node automatically downloads all the required modules, the correct version of each package is used and not some later version that might or might not have bugs that cause conflicts within your application.