The intent of this document is to explain how this login application works.  Users are invited to sign-in and if a member, will allow them to login-in with data authenticated then provided a member page in the browser.  Member usernames and encrypted passwords are stored in a cloud database.  New to the application will require users sign-in.  Data is stored once they sign-in and users are provided the member page.

This program is designed with all folders inside the **DEVELOP DIRECTORY**.  Folders consist of Config, Models, Node Modules, Public, Routes plus the Package.json file.

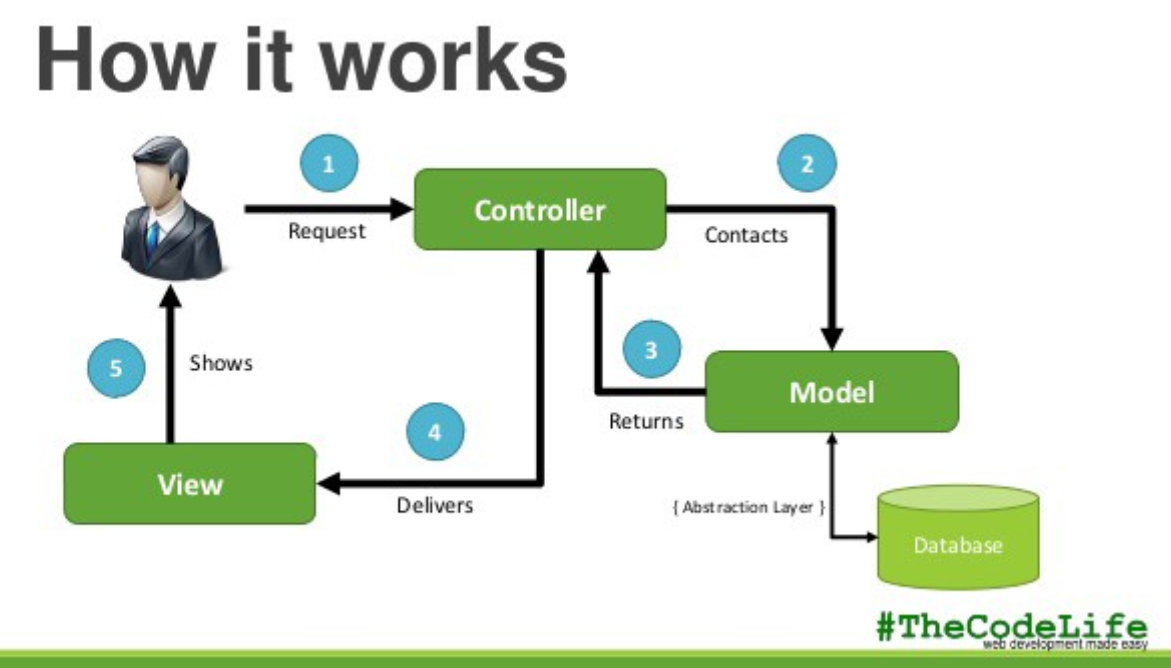
**NPM START** runs package.json scripts. To address the mySQL database, it is necessary to add my personal password in the **Package.json**.

Dependencies include Express, MySQL, Sequelize which connects Express and MySQL.  Additionally bscrypts.js is managing the authentication/encryption and assisting to maintain security.

**Express Session** is needed to login/out of the app and prevent users from logging in with the wrong password.  Authentication and encryption are used to secure data and keeps information hidden.  Bscripts and Passport are needed for encryption and authentication.

Express Session secures the connection.Bscripts, Express Session, and Passport are considered ***middleware.***

Middleware is software that lies between files/layers/applications running and essentially functioning as hidden translation layers. These are useful for enabling communication and data management for distributed applications.

**This document will help explain how information is passed.**

**Simple case explanation:**

New User requests sign-up page that is a request to Controller (1) in the form of HTML route.  Existing User has a username and password stored in the database so they are making the html call then the api router takes them to the Model (2) or JavaScript files that connect to the Database and then return (3) for authentication, thus repeating the pass through the model/database, before delivering the member page through View and accessing the HTML and CSS to render the Member page.

**SERVER.JS**

The **server.js** file uses express to create a server which listens to port 8080

Dependencies are:  “Express”, “Express-session”, “Passport”

This is the main trigger for the application.  In line 2 It first sets up the npm packages then sets up the port and finally the db.  On line 9 in server.js, sets up the main database and calls dependencies inside the Models folder.

Line 12 sets up Express and establishes the language used with the database.  Line 15 calls the ***static*** data in the **PUBLIC** directory.  Line 17-9 sets up the session using middleware.  Line 22 calls items in the **ROUTES** directory and passes “Express”, “Express-session”, “Passport” functions and creates them as global functions.

**Server.js** file is “constructing” until the calls are made to link or sync with the database on 26.

Line 26 is needed for Sequelize once all is configured then runs.  Note:  Written in ES5 using %s.  Note that all statements must be true to pass information.

**PACKAGE.JSON**

Contains the meta information and tells which script to read as well as a list of dependencies to make the application run  **NPM INSTALL** created this file and add dependencies.

In general Package.json dependencies are setup-called then server.js calls Express to connect to the Router.  HTML connections start with the “/” main or root and set up port 8080 to establish connection.  Check if the user meets requirements and invokes authentication with middleware.  Then config files set up with models files and Sequelize creates tables with title and grabs information to create an email, password title by passing strings and validating unique.  Line 22 in **passport.js** validates  and encrypts password for storage.

Passport,js reads and writes into the database.  Passport-local and LocalStrategy only change if no email or email is changed.  This program cannot run unless the email is checked.  Email addresses are used as gate-keepers.  If they are empty or aren’t a match, or !=, errors are thrown.  **User.js** is leverages for password creation, storage, authentication.  Line 40 is worth noting that cd stands for callback.

Passport,js is used to check user ID and password then checks dbuser and verifies it is not empty and a match.  Once complete validates password works.  Sequelize boiler-plates functions are used on line 40 and 44. This file ends with exporting passport which then is used in making api-routes.js and html shows the actual page to the user.  API-routes.js is like a traffic director/enabler always grabbing, deleting, moving as needed to communicate.

**ROUTES DIRECTORY** contains two JavaScript files; **api-routes.js** and **html-routes.js**.

**Api-routes.js** establishes a route connection to the data while **html-routes.js** is like a pipe for the API data.

After lines 2 and 5 set up the “require” path to the HTML files  and middleware for authentication, Line 7 in the **html-routes.js** file is throwing the “module” function and grabbing Express in the “get” function. Line 9 identifies the location “/” which is the root.  Line 12 checks to see if the user has an account and directs them to the “members” page.  Line 14 directs the user to the **signup.html**.  Line 17 directs users to “login” if they are a member.  Line 27 invokes isAuthenticated middleware to assist and redirect users who have not logged in back to the sign-up function.  This is written to establish a global function.

Similarly in the **api-routes.js** file lines 2-3 set up the “require” to configure the database with files in the “models” folder and passport which calls the authentication middleware.

In general any time **CRUD (**Create, Read, Update, Destroy) requires the database and utilizes the **api-route.js** as well as other files.  Anytime you use CRUD, you are connecting to the database.

HTTP status codes are used in api-routes.js file.  Line 22 throws a redirection error while line 25 throws a client error side error.  Here is a list of HTTP Status Codes:

100  info error

200  success

300  redirection

400  client side error

500 server side error

If the user signs out of logs off, 307 code is used to note the redirect.  Login.js executes login with a promise and catch and calls on api-router.js to throw the status codes.

**PUBLIC DIRECTORY** contains two directories; **js** and **stylesheets**.  Additionally there are three html files; **login, members, signup**.  This is where the user interface is handled and public connections are managed.  Again in the **html-routes.js** file line 14 the “sendFile” defines the path to sign-up then pushes to the login if needed while the isAuthenticated middleware checks the database name then asks for sign-in or login.

The HTML and CSS files inside the **PUBLIC** directory are used to set up the user views once they input their initial request in the browser to bring up the application then respond with sign-up, login, or log out.

HTML files consist of **Login.html, Members.html, and Signup.html.**

The **CONFIG DIRECTORY** contains the **config.json** and **passport.js** files plus a **MIDDLEWARE  SUB-DIRECTORY**.  **Config.json** defines connections for “development”, “test”, and “production” by defining their configuration parameters.  Here is where the database language, login and port information is contained.  In this file, “test” protects from hackers, “production” makes sure connection is run in the development mode and connects **Passport.js** sets up the middleware and helps protect from hackers.

**MODELS DIRECTORY** contains three JavaScript files called index.js, product.js and user.js.  Index.js is the primary driving file that first defines the file system, path, Sequelize, the environment, configuration and opens doors for Sequelize to be useful. Line 1 in index.js uses the “strict” rule to ensure assumptions or exceptions are tolerated.  I**ndex.js** is like a MC or traffic director.  This file passes information.   Index.js calls the donfi dev mode line 8 and throws information on line 9 into what is now a clean database.  (**Index.js** is like the broom closet.  The buildings, or in this case the connections and database/tables are cleaned but nobody ever cleans the broom closet.)

In **index.js** line 11 and 12 addresses Heroku and if Heroku is not set-up lines 13-14 will manage by creating Sequelize before calling the database to make sure information is correct or okay. In line 20 (‘.’) means ALL and not zero are true before this runs.

Once **index.js** ensures the database is ready, line 27 calls the Object to have Sequelize create user.js and carry username, password, and encryption.Module.export makes this a global, throwing data

Index.js line 17 creates the table.

Object function on line 27 in **index.js** uses Sequelize and once the database is ready created user.js and carries username, password, and encryption to the database.  Module.export makes this global, throwing data to sequelize and a constructor is used to build data.

Here is a copy of what was displayed when the application was run in the terminal:

$ node server.js

Executing (default): CREATE TABLE IF NOT EXISTS `Products` (`id` INTEGER NOT NULL auto\_increment , `productName` VARCHAR(255) NOT NULL, `categoryType` VARCHAR(255) NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;

Executing (default): SHOW INDEX FROM `Products` FROM `passport\_demo`

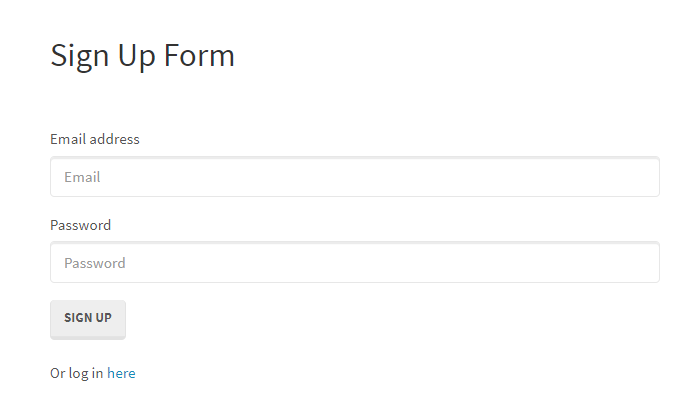
Executing (default): CREATE TABLE IF NOT EXISTS `Users` (`id` INTEGER NOT NULL auto\_increment , `email` VARCHAR(255) NOT NULL UNIQUE,

`password` VARCHAR(255) NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;

Executing (default): SHOW INDEX FROM `Users` FROM `passport\_demo`

==> � Today is the day!  Listening on port 8080. Visit http://localhost:8080/ in your browser.

Here is a screen shot of the application which can be loaded in the browser using the last line above:



Here is the message received once successfully logged in:

