

# Reverse Engineering Code

## Tutorial for a Node.js application using Sequelize and Passport

By Henry Jean Logique

This tutorial is a walk-through of the codebase for developers. Its main goal is to inspect codes, and explain every file and its purpose in the application, which can be used as a starting point for a new project. This application allows a user to signup and create an account, log into an existing account, and logout securely. User credentials are stored in a MySQL database.

## USER STORY

As a user I want to securely create a new account on a site and sign in to it, and my account information is securely and correctly stored in a database.

## TABLE OF CONTENT

[FILE DIRECTORY MAP](#)

[INSTALLATION AND USAGE](#)

[FILES WALK-THROUGH](#)

[package.json](#)

[server.js](#)

[isAuthenticated.js](#)

[config.json](#)

[passport.js](#)

[index.js](#)

[user.js](#)

[login.js](#)

[members.js](#)

[signup.js](#)

[style.css](#)

[login.html](#)

[members.html](#)

[signup.html](#)

[api-routes.js](#)  
[html-routes.js](#)  
[schema.sql](#)  
[package-lock.json](#)  
[node\\_modules](#)

[What to Test?](#)

[Test Cases:](#)

[Test Environments:](#)

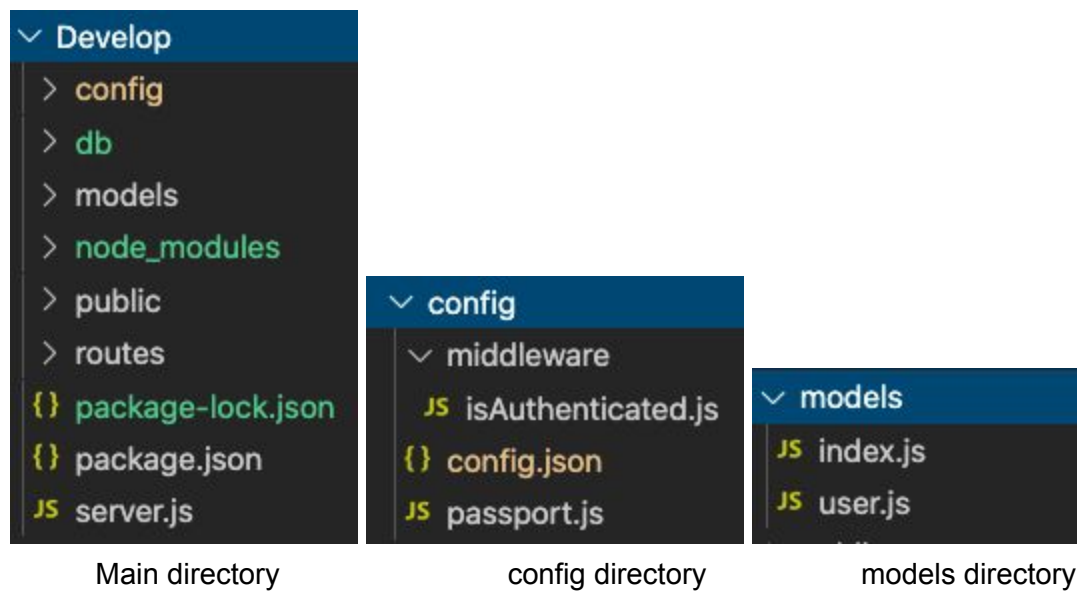
[Testing Tools:](#)

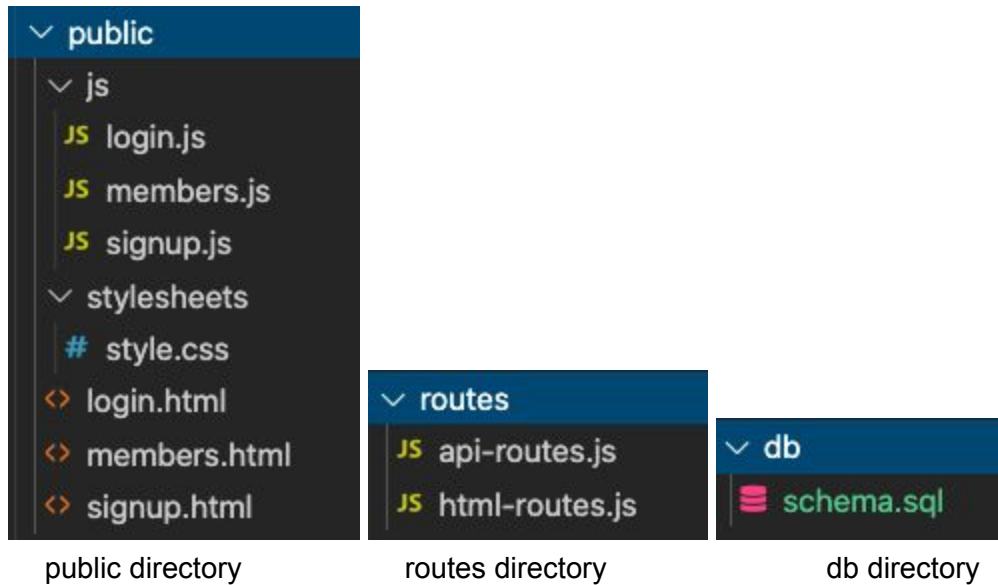
[What's next?](#)

[Resources](#)

## FILE DIRECTORY MAP

The **Develop** (main or root) directory contains the following sub-directories:





## INSTALLATION AND USAGE

- First clone the repository, or download the **GitHub** package to your local machine.
- Add your MySQL password to the **config.json** file in the config directory.

```
{
  "development": {
    "username": "root",
    "password": "",
    "database": "passport_demo",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "test": {
    "username": "root",
    "password": "",
    "database": "database_test",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "production": {
    "username": "root",
    "password": "",
    "database": "database_production",
    "host": "127.0.0.1",
    "dialect": "mysql"
  }
}
```

- Use the **schema.sql** file to create three MySQL databases: **passport\_demo**, **database\_test** and **database\_production**, as shown below. These databases are used in the **config.json** file, as shown in the above image.

```
1  -- SQL Code to create database, name was specified in the config.json file
2  -- Drops the database if it exists currently --
3  DROP DATABASE IF EXISTS database_production;
4  DROP DATABASE IF EXISTS database_test;
5  DROP DATABASE IF EXISTS passport_demo;
6
7  -- Creates the "database" database --
8  CREATE DATABASE database_production;
9  CREATE DATABASE database_test;
10 CREATE DATABASE passport_demo;
```

You can copy and paste the above MySQL statements to your MySQL editor or execute the **schema.sql** file in the integrated terminal of the root directory to create the databases:

**mysql -uroot -p < schema.sql**

- Next install the dependencies by typing **npm i** in the integrated terminal of the root (here **Develop**) directory.

**Note:** The seven dependencies are listed in the **package.json** file, as shown below.

```
"dependencies": {
  "bcryptjs": "2.4.3",
  "express": "^4.17.0",
  "express-session": "^1.16.1",
  "mysql2": "^1.6.5",
  "passport": "^0.4.0",
  "passport-local": "^1.0.0",
  "sequelize": "^5.8.6"
}
```

- To run the application, type node **server.js** in the integrated terminal of the root directory.
- Open the browser with the URL: **localhost:8080**

localhost:8080

## Sign Up Form

Email address

Password

**SIGN UP**

Or log in [here](#)

- **Sign up** with a new email and a password
- You automatically go to the **members** page, as shown below.

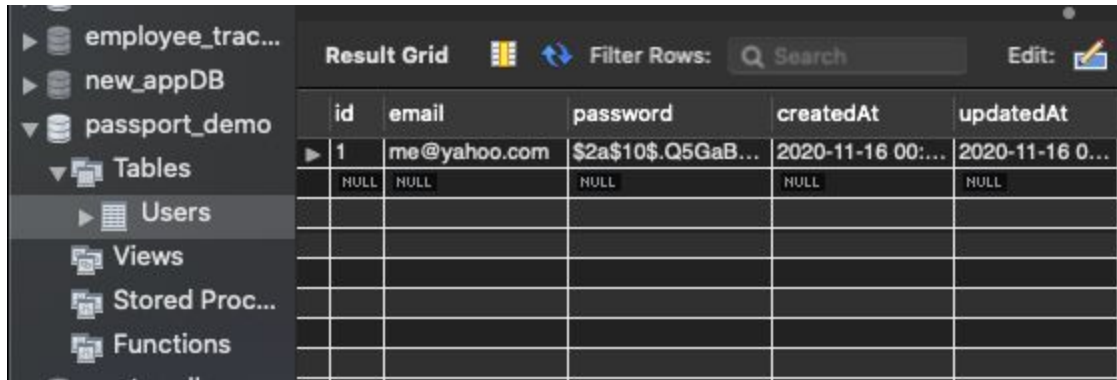
localhost:8080/members

[Logout](#)

Welcome me@yahoo.com

- Sign out by selecting the **Logout** link.
- Select the **log in here** link to go to the login page, and sign in with your email and password

- Check the **passport\_demo** database on your visual database design tool (here **MySQL Workbench**). The following image shows the new created account (row) in the **Users** table of the **passport\_demo** database.



The screenshot shows the MySQL Workbench interface. On the left, the 'passport\_demo' database is selected, and the 'Users' table is highlighted under the 'Tables' section. The main area displays the 'Result Grid' for the 'Users' table. The table has five columns: 'id', 'email', 'password', 'createdAt', and 'updatedAt'. The first row contains the data for a newly created user: '1', 'me@yahoo.com', '\$2a\$10\$.Q5GaB...', '2020-11-16 00:...', and '2020-11-16 0...'. Below this row, there are several rows with 'NULL' values, indicating that the table is currently empty except for the one user.

id	email	password	createdAt	updatedAt
1	me@yahoo.com	\$2a\$10\$.Q5GaB...	2020-11-16 00:...	2020-11-16 0...
NULL	NULL	NULL	NULL	NULL

## FILES WALK-THROUGH

### package.json

(path: Develop/package.json)

All **npm** packages contain this file, usually in the project root. It holds various **metadata** and is used to give information to npm that allows it to identify the project as well as handles the project's **dependencies**. It can also contain other metadata such as a project **description**, **version**, **license** information, even **configuration data**. The listed dependencies are used by the following files:

- server.js
- config/passport.js
- models/index.js
- models/user.js

```
{
  "name": "1-Passport-Example",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js",
    "watch": "nodemon server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "2.4.3",
    "express": "^4.17.0",
    "express-session": "^1.16.1",
    "mysql2": "^1.6.5",
    "passport": "^0.4.0",
    "passport-local": "^1.0.0",
    "sequelize": "^5.8.6"
  }
}
```

## server.js

(path: Develop/server.js)

This file

- requires **express** and **express-session** packages, and the configured **passport** (**passport.js**) once the **npm** project's **dependencies** listed in the **package.json** file are installed.
- sets up **PORT** using the **environment variable** or **8080**.
- creates the **express** server.
- configures built-in **middleware** functions in **express** server.
- keeps track of the user's login status by using **sessions** and methods of the **passport**.
- creates **html** and **api** routes by requiring the **html\_routes.js** and **api\_routes.js**.
- syncs database: creates tables according to the schema specified in the model.
- logs a message in the terminal once the connection to the server is successfully established.

```
// Requiring necessary npm packages
var express = require("express");
var session = require("express-session");
// Requiring passport as we've configured it
var passport = require("../config/passport");

// Setting up port and requiring models for syncing
var PORT = process.env.PORT || 8080;
var db = require("../models");

// Creating express app and configuring middleware needed for authentication
var app = express();
app.use(express.urlencoded({ extended: true }));
app.use(express.json());
app.use(express.static("public"));
// We need to use sessions to keep track of our user's login status
app.use(session({ secret: "keyboard cat", resave: true, saveUninitialized: true }));
app.use(passport.initialize());
app.use(passport.session());

// Requiring our routes
require("../routes/html-routes.js")(app);
require("../routes/api-routes.js")(app);

// Syncing our database and logging a message to the user upon success
db.sequelize.sync().then(function() {
  app.listen(PORT, function() {
    console.log("=> 🌐 Listening on port %s. Visit http://localhost:%s/ in your browser.", PORT, PORT);
  });
});
```



## isAuthenticated.js *(path: Develop/config/middleware/isAuthenticated.js)*

This is a custom middleware for restricting routes users are not allowed to visit. If users who are not logged in try to access the restricted route (members page), they are redirected to the signup page. Whereas, if they are logged in, they can continue with the request to the restricted route. The **html-routes.js** file uses this middleware for authentication.

```
// This is middleware for restricting routes a user is not allowed to visit if not logged in
module.exports = function(req, res, next) {
  // If the user is logged in, continue with the request to the restricted route
  if (req.user) {
    return next();
  }

  // If the user isn't logged in, redirect them to the login page
  return res.redirect("/");
};
```

## config.json *(path: Develop/config/config.json)*

This file is used by **Sequelize** to manage different environments, and access databases. It includes the login information for databases, names of the databases, their locations and their types. The **index.js** file requires it.

```
{
  "development": {
    "username": "root",
    "password": "",
    "database": "passport_demo",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "test": {
    "username": "root",
    "password": "",
    "database": "database_test",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "production": {
    "username": "root",
    "password": "",
    "database": "database_production",
    "host": "127.0.0.1",
    "dialect": "mysql"
  }
}
```



## passport.js

(path: *Develop/config/passport.js*)

This file is an authentication middleware (allowing users to log in), which works with express, and allows for different authentication strategies. The Local Strategy allows us to authenticate users by looking up their data (emails and passwords) in the database.

There are three main parts in using passport.js:

- Requiring the module and using its `passport.initialize()` and `passport.session()` middleware with express.
- Configuring passport with at least one Strategy and setting up passport's `serializeUser` and `deserializeUser` methods.
- Specifying a route which uses the `passport.authenticate` middleware to actually authenticate a user.

The **server.js** and **api\_route.js** require the **passport**.

```
1  var passport = require("passport");
2  var LocalStrategy = require("passport-local").Strategy;
3
4  var db = require("../models");
5
6  // Telling passport we want to use a Local Strategy. In other words, we want login with a username/email and password
7  passport.use(new LocalStrategy(
8    // Our user will sign in using an email, rather than a "username"
9    {
10     usernameField: "email"
11   },
12   function(email, password, done) {
13     // When a user tries to sign in this code runs
14     db.User.findOne({
15       where: {
16         email: email
17       }
18     }).then(function(dbUser) {
19       // If there's no user with the given email
20       if (!dbUser) {
21         return done(null, false, {
22           message: "Incorrect email."
23         });
24       }
25       // If there is a user with the given email, but the password the user gives us is incorrect
26       else if (!dbUser.validPassword(password)) {
27         return done(null, false, {
28           message: "Incorrect password."
29         });
30       }
31       // If none of the above, return the user
32       return done(null, dbUser);
33     });
34   }
35 ));
36
37 // In order to help keep authentication state across HTTP requests,
38 // Sequelize needs to serialize and deserialize the user
39 // Just consider this part boilerplate needed to make it all work
40 passport.serializeUser(function(user, cb) {
41   cb(null, user);
42 });
43
44 passport.deserializeUser(function(obj, cb) {
45   cb(null, obj);
46 });
47
48 // Exporting our configured passport
49 module.exports = passport;
```

## index.js

(path: Develop/models/index.js)

This file is generated with the **sequelize CLI** and collects all the models from the **models** directory and associates them if needed. In other words, **Sequelize** looks for this file to create instances of the models in the **models** directory, and syncs the models with the database and afterwards starts the server. **Index.js** connects to the database and imports each user's login data. The **server.js** and **passport.js** require this model.

```
1  'use strict';
2
3  var fs      = require('fs');
4  var path    = require('path');
5  var Sequelize = require('sequelize');
6  var basename = path.basename(module.filename);
7  var env      = process.env.NODE_ENV || 'development';
8  var config   = require(__dirname + '/../config/config.json')[env];
9  var db       = {};
10
11  if (config.use_env_variable) {
12    var sequelize = new Sequelize(process.env[config.use_env_variable]);
13  } else {
14    var sequelize = new Sequelize(config.database, config.username, config.password, config);
15  }
16
17  fs
18    .readdirSync(__dirname)
19    .filter(function(file) {
20      return (file.indexOf('.') !== 0) && (file !== basename) && (file.slice(-3) === '.js');
21    })
22    .forEach(function(file) {
23      var model = sequelize['import'](path.join(__dirname, file));
24      db[model.name] = model;
25    });
26
27  Object.keys(db).forEach(function(modelName) {
28    if (db[modelName].associate) {
29      db[modelName].associate(db);
30    }
31  });
32
33  db.sequelize = sequelize;
34  db.Sequelize = Sequelize;
35
36  module.exports = db;
```

## user.js

(path: *Develop/models/user.js*)

This is another model of the application, which inserts user information to the database using **sequelize**, and represents the **User** table in the database. It verifies that the email is not null and valid, and uses the **bcryptjs** to hash passwords before users are created. It checks if an unhashed password entered by users can be compared to the hashed passwords stored in the database. In other words, it checks if a user already exists in the database, and if not it adds the user. The **server.js** and **passport.js** require this model.

```
1 // Requiring bcrypt for password hashing. Using the bcryptjs version as the
2 // regular bcrypt module sometimes causes errors on Windows machines
3 var bcrypt = require("bcryptjs");
4 // Creating our User model
5 module.exports = function(sequelize, DataTypes) {
6   var User = sequelize.define("User", {
7     // The email cannot be null, and must be a proper email before creation
8     email: {
9       type: DataTypes.STRING,
10      allowNull: false,
11      unique: true,
12      validate: {
13        isEmail: true
14      }
15    },
16    // The password cannot be null
17    password: {
18      type: DataTypes.STRING,
19      allowNull: false
20    }
21  });
22  // Creating a custom method for our User model. This will check if an unhashed password
23  // entered by the user can be compared to the hashed password stored in our database
24  User.prototype.validPassword = function(password) {
25    return bcrypt.compareSync(password, this.password);
26  };
27  // Hooks are automatic methods that run during various phases of the User Model lifecycle
28  // In this case, before a User is created, we will automatically hash their password
29  User.addHook("beforeCreate", function(user) {
30    user.password = bcrypt.hashSync(user.password, bcrypt.genSaltSync(10), null);
31  });
32  return User;
33 };
```

## login.js

(path: Develop/public/js/login.js)

This file together with other html and css files in the **public** directory create the user interface (UI). It allows capturing the user's information for logging in. This file handles the form validation and submission in the **login.html** file, and redirects valid members to the members page. It is used by the **api-routes.js**.

```
1 $(document).ready(function() {
2     // Getting references to our form and inputs
3     var loginForm = $("form.login");
4     var emailInput = $("input#email-input");
5     var passwordInput = $("input#password-input");
6
7     // When the form is submitted, we validate there's an email and password entered
8     loginForm.on("submit", function(event) {
9         event.preventDefault();
10        var userData = {
11            email: emailInput.val().trim(),
12            password: passwordInput.val().trim()
13        };
14
15        if (!userData.email || !userData.password) {
16            return;
17        }
18
19        // If we have an email and password we run the loginUser function and clear the form
20        loginUser(userData.email, userData.password);
21        emailInput.val("");
22        passwordInput.val("");
23    });
24
25    // loginUser does a post to our "api/login" route and if successful, redirects us the the members page
26    function loginUser(email, password) {
27        $.post("/api/login", {
28            email: email,
29            password: password
30        })
31        .then(function() {
32            window.location.replace("/members");
33            // If there's an error, log the error
34        })
35        .catch(function(err) {
36            console.log(err);
37        });
38    }
39 });
40
```

## members.js

(path: Develop/public/js/members.js)

This file simply updates the **members.html** file based on the member's login information. It is used by the **api-routes.js**.

```
1 $(document).ready(function() {
2     // This file just does a GET request to figure out which user is logged in
3     // and updates the HTML on the page
4     $.get("/api/user_data").then(function(data) {
5         $(".member-name").text(data.email);
6     });
7 });
8
```



## signup.js

(path: *Develop/public/js/signup.js*)

This file handles the form validation and submission in the **signup.html** file. If a new user account is successfully created, it redirects the new member to the members page. It is used by the **api-routes.js**.

```
1  $(document).ready(function() {
2      // Getting references to our form and input
3      var signUpForm = $("form.signup");
4      var emailInput = $("input#email-input");
5      var passwordInput = $("input#password-input");
6
7      // When the signup button is clicked, we validate the email and password are not blank
8      signUpForm.on("submit", function(event) {
9          event.preventDefault();
10         var userData = {
11             email: emailInput.val().trim(),
12             password: passwordInput.val().trim()
13         };
14
15         if (!userData.email || !userData.password) {
16             return;
17         }
18         // If we have an email and password, run the signUpUser function
19         signUpUser(userData.email, userData.password);
20         emailInput.val("");
21         passwordInput.val("");
22     });
23
24     // Does a post to the signup route. If successful, we are redirected to the members page
25     // Otherwise we log any errors
26     function signUpUser(email, password) {
27         $.post("/api/signup", {
28             email: email,
29             password: password
30         })
31         .then(function(data) {
32             window.location.replace("/members");
33             // If there's an error, handle it by throwing up a bootstrap alert
34         })
35         .catch(handleLoginErr);
36     }
37
38     function handleLoginErr(err) {
39         $("#alert .msg").text(err.responseJSON);
40         $("#alert").fadeIn(500);
41     }
42 });
```

## style.css

(path: Develop/public/stylesheets/style.css)

This file is used by **login.html** and **signup.html**.

```
1  form.signup,
2  form.login {
3    margin-top: 50px;
4  }
5
```

## login.html

(path: Develop/public/login.html)

This is the login web page of the application. It contains a form which is handled by **login.js**. This file is used by **html-routes.js**.

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <title>Passport Authentication</title>
6    <meta charset="UTF-8">
7    <meta name="viewport" content="width=device-width, initial-scale=1">
8    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/lumen/bootstrap.min.css">
9    <link href="stylesheets/style.css" rel="stylesheet">
10 </head>
11
12 <body>
13   <nav class="navbar navbar-default">
14     <div class="container-fluid">
15       <div class="navbar-header">
16       </div>
17     </div>
18   </nav>
19   <div class="container">
20     <div class="row">
21       <div class="col-md-6 col-md-offset-3">
22         <h2>Login Form</h2>
23         <form class="login">
24           <div class="form-group">
25             <label for="exampleInputEmail1">Email address</label>
26             <input type="email" class="form-control" id="email-input" placeholder="Email">
27           </div>
28           <div class="form-group">
29             <label for="exampleInputPassword1">Password</label>
30             <input type="password" class="form-control" id="password-input" placeholder="Password">
31           </div>
32           <button type="submit" class="btn btn-default">Login</button>
33         </form>
34         <br />
35         <p>Or sign up <a href="/">here</a></p>
36       </div>
37     </div>
38   </div>
39
40   <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
41   <script type="text/javascript" src="js/login.js"></script>
42
43 </body>
44
45 </html>
```

## members.html

(path: Develop/public/members.html)

This is the members web page, which is updated by **members.js**. It contains the **logout** link. This file is used by **html-routes.js**.

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <title>Passport Authentication</title>
6    <meta charset="UTF-8">
7    <meta name="viewport" content="width=device-width, initial-scale=1">
8    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootswatch/3.3.7/lumen/bootstrap.min.css">
9    <link href="stylesheets/style.css" rel="stylesheet">
10 </head>
11
12 <body>
13   <nav class="navbar navbar-default">
14     <div class="container-fluid">
15       <div class="navbar-header">
16         <a class="navbar-brand" href="/logout">
17           Logout
18         </a>
19       </div>
20     </div>
21   </nav>
22   <div class="container">
23     <div class="row">
24       <div class="col-md-6 col-md-offset-3">
25         <h2>Welcome <span class="member-name"></span></h2>
26       </div>
27     </div>
28   </div>
29
30   <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
31   <script type="text/javascript" src="js/members.js"></script>
32
33 </body>
34
35 </html>
```



## signup.html

(path: Develop/public/signup.html)

This is the signup web page of the application. It contains a form which is handled by **signup.js**. This file is used by **html-routes.js**.

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <title>Passport Authentication</title>
6    <meta charset="UTF-8">
7    <meta name="viewport" content="width=device-width, initial-scale=1">
8    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/lumen/bootstrap.min.css">
9    <link href="stylesheets/style.css" rel="stylesheet">
10 </head>
11
12 <body>
13   <nav class="navbar navbar-default">
14     <div class="container-fluid">
15       <div class="navbar-header">
16       </div>
17     </div>
18   </nav>
19   <div class="container">
20     <div class="row">
21       <div class="col-md-6 col-md-offset-3">
22         <h2>Sign Up Form</h2>
23         <form class="signup">
24           <div class="form-group">
25             <label for="exampleInputEmail">Email address</label>
26             <input type="email" class="form-control" id="email-input" placeholder="Email">
27           </div>
28           <div class="form-group">
29             <label for="exampleInputPassword1">Password</label>
30             <input type="password" class="form-control" id="password-input" placeholder="Password">
31           </div>
32           <div style="display: none" id="alert" class="alert alert-danger" role="alert">
33             <span class="glyphicon glyphicon-exclamation-sign" aria-hidden="true"></span>
34             <span class="sr-only">Error:</span> <span class="msg"></span>
35           </div>
36           <button type="submit" class="btn btn-default">Sign Up</button>
37         </form>
38         <br />
39         <p>Or log in <a href="/login">here</a></p>
40       </div>
41     </div>
42   </div>
43
44   <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
45   <script type="text/javascript" src="js/signup.js"></script>
46
47 </body>
48
49 </html>
```

## api-routes.js

(path: Develop/routes/api-routes.js)

The file is used to route **GET** and **POST** from and to the database. It requires **Sequelize models** and **passport** to automatically hash passwords, validate and securely store users login credentials, and redirect users to the **members** page. It has different routes for logging in, signing up, logging out and getting some data about users to be used on the client side. It is used by **server.js**.

```
1 // Requiring our models and passport as we've configured it
2 var db = require("../models");
3 var passport = require("../config/passport");
4
5 module.exports = function(app) {
6   // Using the passport.authenticate middleware with our local strategy.
7   // If the user has valid login credentials, send them to the members page.
8   // Otherwise the user will be sent an error
9   app.post("/api/login", passport.authenticate("local"), function(req, res) {
10     res.json(req.user);
11   });
12
13   // Route for signing up a user. The user's password is automatically hashed and stored securely thanks to
14   // how we configured our Sequelize User Model. If the user is created successfully, proceed to log the user in,
15   // otherwise send back an error
16   app.post("/api/signup", function(req, res) {
17     db.User.create({
18       email: req.body.email,
19       password: req.body.password
20     })
21     .then(function() {
22       res.redirect(307, "/api/login");
23     })
24     .catch(function(err) {
25       res.status(401).json(err);
26     });
27   });
28
29   // Route for logging user out
30   app.get("/logout", function(req, res) {
31     req.logout();
32     res.redirect("/");
33   });
34
35   // Route for getting some data about our user to be used client side
36   app.get("/api/user_data", function(req, res) {
37     if (!req.user) {
38       // The user is not logged in, send back an empty object
39       res.json({});
40     } else {
41       // Otherwise send back the user's email and id
42       // Sending back a password, even a hashed password, isn't a good idea
43       res.json({
44         email: req.user.email,
45         id: req.user.id
46       });
47     }
48   });
49 };
```

## html-routes.js

(path: Develop/routes/html-routes.js)

It requires **path** to use relative routes to the **login.html**, **signup.html** and **members.html** files, and requires the **isAuthenticated** middleware to check if users are logged in to send them to the members page. It has a route to check if users who are not logged in try to access the members page, and redirect them to the signup page. It is used by **server.js**.

```
1  // Requiring path to so we can use relative routes to our HTML files
2  var path = require("path");
3
4  // Requiring our custom middleware for checking if a user is logged in
5  var isAuthenticated = require("../config/middleware/isAuthenticated");
6
7  module.exports = function(app) {
8
9    app.get("/", function(req, res) {
10      // If the user already has an account send them to the members page
11      if (req.user) {
12        res.redirect("/members");
13      }
14      res.sendFile(path.join(__dirname, "../public/signup.html"));
15    });
16
17    app.get("/login", function(req, res) {
18      // If the user already has an account send them to the members page
19      if (req.user) {
20        res.redirect("/members");
21      }
22      res.sendFile(path.join(__dirname, "../public/login.html"));
23    });
24
25    // Here we've add our isAuthenticated middleware to this route.
26    // If a user who is not logged in tries to access this route they will be redirected to the signup page
27    app.get("/members", isAuthenticated, function(req, res) {
28      res.sendFile(path.join(__dirname, "../public/members.html"));
29    });
30
31  };
```

## schema.sql

(path: Develop/db/schema.sql)

This file creates the databases. If they already exist, it remove and recreate them.

```
1  -- SQL Code to create database, name was specified in the config.json file
2  -- Drops the database if it exists currently --
3  DROP DATABASE IF EXISTS database_production;
4  DROP DATABASE IF EXISTS database_test;
5  DROP DATABASE IF EXISTS passport_demo;
6
7  -- Creates the "database" database --
8  CREATE DATABASE database_production;
9  CREATE DATABASE database_test;
10 CREATE DATABASE passport_demo;
```

## package-lock.json

(path: Develop/package-lock.json)

This file is used to lock dependencies to a specific version number.

**Note:** This is a very large file. The image below only displays a part of the file.

```
1  {
2    "name": "1-Passport-Example",
3    "version": "1.0.0",
4    "lockfileVersion": 1,
5    "requires": true,
6    "dependencies": {
7      "@types/node": {
8        "version": "14.14.7",
9        "resolved": "https://registry.npmjs.org/@types/node/-/node-14.14.7.tgz",
10       "integrity": "sha512-Zw1vhUSQZYw+7u5dAwNbIA9TuTotpzY/0F7sJM9FqP0F3SPjKnxrjoTktXDZgUjybf4cwVBP708wvKdSaGHweg==",
11      },
12      "accepts": {
13        "version": "1.3.7",
14        "resolved": "https://registry.npmjs.org/accepts/-/accepts-1.3.7.tgz",
15        "integrity": "sha512-I80Qs2WjYlJIBNzNkK6KYqlVMTbZLXgHx2oT0pU/fjRHyEp+PEfEPY0R3WCwAGV0tauxh1h0xNgIf5bv7dQpA==",
16        "requires": {
17          "mime-types": "~2.1.24",
18          "negotiator": "0.6.2"
19        }
20      },
21      "any-promise": {
22        "version": "1.3.0",
23        "resolved": "https://registry.npmjs.org/any-promise/-/any-promise-1.3.0.tgz",
24        "integrity": "sha1-q8av7tzqUugJzcA3au0845Y10X8=",
25      },
26      "array-flatten": {
27        "version": "1.1.1",
28        "resolved": "https://registry.npmjs.org/array-flatten/-/array-flatten-1.1.1.tgz",
29        "integrity": "sha1-m19pkF6x5wcZKPKgCJaLZ0opVdI=",
30      },
31      "bcryptjs": {
32        "version": "2.4.3",
33        "resolved": "https://registry.npmjs.org/bcryptjs/-/bcryptjs-2.4.3.tgz",
34        "integrity": "sha1-mrVie5PmBiH/fNrF2pczAn3x0Ms=",
35      },
36      "bluebird": {
37        "version": "3.7.2",
38        "resolved": "https://registry.npmjs.org/bluebird/-/bluebird-3.7.2.tgz",
39        "integrity": "sha512-XpNj6GDQzdfW+r2Wnn7xiSAd7TM3jzkkXBGttWkuSXv1xUV+azxAm8jdWZN06QTQk+2N2XB9jRDkvbMqmcRtg==",
40      },
41      "body-parser": {
42        "version": "1.19.0",
43        "resolved": "https://registry.npmjs.org/body-parser/-/body-parser-1.19.0.tgz",
44        "integrity": "sha512-dhEPs72UPbDnAQJ9ZKMNTP6ptJai0nhP5cBb541nXPLW60Jepo9RV/a4fX4XWw9CuFNk22krhrj1+rgzifNCsw==",
45        "requires": {
46          "bytes": "3.1.0",
47          "content-type": "~1.0.4",
48          "debug": "2.6.9",
49          "depd": "~1.1.2",
```

## node\_modules

(path: Develop/node\_modules)

Once the **npm** dependencies are installed locally, the required packages are dropped into the **node\_modules** directory. This means that **require("packagename")** loads its main module.

# What to Test?

## Test Cases:

- Sign up a new account, you should be redirected to the members page. Logout, you should be successfully signed out, and not be able to return to the members page without valid credentials. Check the database for your credentials.
- Login with valid credentials, you should be redirected to the members page.
- Try to login with invalid credentials, you should not be able to go to the members page. You should receive a valid error message
- Try to go to the members page without signing in, you should be redirected to the signup page.
- Try to sign up with no entries, you should be received a valid error message and no account should be created. Check the database for null values.

## Test Environments:

- Integrated Terminal
- Application User Interface
- Database

## Testing Tools:

- Postman
- Jest
- Console log

## What's next?

What to change to improve this application?

- Apply **validations** for existing passwords and emails in the database, and replace throwing exceptions and errors with meaningful **warnings**.
- Apply validations for passwords and emails, which do not meet the requirements, and replace throwing exceptions and errors with meaningful **warnings**.

- Add a section for **changing emails and passwords** to the application.
- Add **Forgot Username and Password** functionality to the application.
- Apply **ES6** features like **let**, **const**, **arrow functions** to the code.
- Simplify the code with **.map()**, **.filter()** and **.reduce()**.
- Modify **style.css** file to improve the looks and designs of the html files.
- Integrate the application into a website to use its features to secure the site.
- Instead of hosting the application locally, host it on the Heroku server with Heroku MySql database.

## Resources

<https://sequelize.org/master/manual/querying.html>

<https://www.npmjs.com/package/bcrypt>

<http://www.passportjs.org/docs/configure/>

<https://dev.to/darshanbib/user-management-for-node-js-mysql-using-sequelize-and-passportjs-44kj>

<https://code.tutsplus.com/tutorials/using-passport-with-sequelize-and-mysql--cms-27537>

<https://jasonwatmore.com/post/2020/08/18/nodejs-mysql-simple-api-for-authentication-registration-and-user-management>

<https://developer.okta.com/blog/2018/06/28/tutorial-build-a-basic-crud-app-with-node>

<https://sequelize.readthedocs.io/en/1.7.0/articles/express/>

<http://toon.io/understanding-passportjs-authentication-flow/>

<https://medium.com/@saranyamohandas2/passportjs-with-mysql-sequelize-and-expressjs-56fb903aaf8f>

<https://scotch.io/@GM456742/building-a-nodejs-web-app-using-passportjs-for-authentication>

<https://medium.com/poka-techblog/simplify-your-javascript-use-map-reduce-and-filter-bd02c593cc2d>