1. The Develop folder holds all the development.  The config folder holds the initial settings for the program.

Config folder holds all the files below:

2. The middleware is a folder within the config folder. It lays between the operating system and the applications running on it. In this folder the middleware is limiting the routes a user will not be able to visit.

3. The file isAuthenticated.js is added right under the middleware.

if (req.user) {

return next();

  }

If the user is logged in then the restricted route will continue.

return res.redirect("/");

If the user is not logged in, then they will be redirected to the login page.

4. Config.json file holds development, test and production information. The user has to connect the MySQL workbench to this file in order for it to run and connect.

5. Passport.js shows how to use its database functions.

passport.use(new LocalStrategy(

The function above is telling passport to use login with username, email, and password.

{

   usernameField: "email"

 },

 function(email, password, done)

)

Above shows users will sign in using their email.

ddb.User.findOne({

     where: {

       email: email

     }

   }).then(function(dbUser) {

The above code will run when a user tries to log in.

if (!dbUser) {

       return done(null, false, {

         message: "Incorrect email."

       });

     }

The above code will run and a message will appear “Incorrect email” if a user doesn’t have a given email.

}

     // If there is a user with the given email, but the password the user gives us is incorrect

     else if (!dbUser.validPassword(password)) {

       return done(null, false, {

         message: "Incorrect password."

       });

     }

The code above will run if an user has a given email, however the user enters the incorrect password.

 return done(null, dbUser);

   });

The code above will run if the user doesn’t have an asigned email and or an incorrect password.

passport.serializeUser(function(user, cb) {

 cb(null, user);

});

passport.deserializeUser(function(obj, cb) {

 cb(null, obj);

});

The above code is showing to keep authentication across HTTP requests, this part needs to work in order to make it all work together.

module.exports = passport;

The code above is showing the converted and the arranged passport.

6. Models folder holds the partial classes.

Models folder holds all the files below:

7. Index.js holds the app start up, this file is also used for easy points for functionality and requiring a module.

8. User.js

var bcrypt = require("bcryptjs");

The above code shows that bcrypt is used for password generating.

module.exports = function(sequelize, DataTypes) {

 var User = sequelize.define("User", {

Here the user model has been created.  
 email: {

     type: DataTypes.STRING,

     allowNull: false,

     unique: true,

     validate: {

       isEmail: true

     }

   },

Must have a proper email before the creation. The email can not be blank (Null).

password: {

     type: DataTypes.STRING,

     allowNull: false

   }

The password can not be blank (Null).

User.prototype.validPassword = function(password) {

   return bcrypt.compareSync(password, this.password);

 };

This is showing a creation of a custom method for the user model.

User.addHook("beforeCreate", function(user) {

   user.password = bcrypt.hashSync(user.password, bcrypt.genSaltSync(10), null);

 });

 return User;

};

The above is showing that automatic passwords will be created even before a user is created.

9. Public folder holds the login, members, and signup js. Public folder

Public folder holds all files below:

10.  Js holds the login.js

var loginForm = $("form.login");

 var emailInput = $("input#email-input");

 var passwordInput = $("input#password-input");

This is showing the references to the form and inputs.

loginForm.on("submit", function(event) {

   event.preventDefault();

   var userData = {

     email: emailInput.val().trim(),

     password: passwordInput.val().trim()

   };

if (!userData.email || !userData.password) {

     return;

   }

This portion is showing that when the form is submitted

loginUser(userData.email, userData.password);

   emailInput.val("");

   passwordInput.val("");

 });

This is showing when an email and password is collected then the loginUser function will run and clear the form.

 function loginUser(email, password) {

   $.post("/api/login", {

     email: email,

     password: password

   })

     .then(function() {

       window.location.replace("/members");

The loginUser creates a post to the ap/login route and if this works then the user will get directed to the members page.

})

     .catch(function(err) {

       console.log(err);

     });

If there is an error created then the error will be logged.

11. Members.js :

$.get("/api/user\_data").then(function(data) {

   $(".member-name").text(data.email);

 });

});

This is showing a GET request to find out which user is logged in, this will then update the HTML.

12. Signup.html:

 var signUpForm = $("form.signup");

 var emailInput = $("input#email-input");

 var passwordInput = $("input#password-input");

Shows references to form and input.

signUpForm.on("submit", function(event) {

   event.preventDefault();

   var userData = {

     email: emailInput.val().trim(),

     password: passwordInput.val().trim()

   };

if (!userData.email || !userData.password) {

     return;

   }

Shows, when a button is clicked the email and password is validated.

signUpUser(userData.email, userData.password);

   emailInput.val("");

   passwordInput.val("");

 });

If there is an email and password then the signUpUser function will run.

function signUpUser(email, password) {

   $.post("/api/signup", {

     email: email,

     password: password

   })

.then(function(data) {

       window.location.replace("/members");

A post is created to the signup route, if this works then will get routed to the members page.   
 })

     .catch(handleLoginErr);

 }

    .catch(handleLoginErr);

 }

 function handleLoginErr(err) {

   $("#alert .msg").text(err.responseJSON);

   $("#alert").fadeIn(500);

 }

});

If there is an error then it will be signaled by creating a bootstrap alert.

13. Style.css references the style for the web page.

14. Login.html- design for login html.

15. Members.html- design for members html.

16. Signup.html- design for singup html.

14.  Routes folder holds mapping and handler information.

Routes folder holds the files below:

15.  Api-routes.js

var db = require("../models");

var passport = require("../config/passport");

module.exports = function(app) {

Shows models and passports.

app.post("/api/login", passport.authenticate("local"), function(req, res) {

   res.json(req.user);

 });

If user has proper logins then they will be sent to the members page.

app.post("/api/signup", function(req, res) {

   db.User.create({

     email: req.body.email,

     password: req.body.password

   })

     .then(function() {

       res.redirect(307, "/api/login");

     })

     .catch(function(err) {

       res.status(401).json(err);

     });

 });

If the user is created properly then the user can proceed to log in.

app.get("/logout", function(req, res) {

   req.logout();

   res.redirect("/");

 });

This shows the route about the user in order to be used on the client’s side.

app.get("/api/user\_data", function(req, res) {

   if (!req.user) {

res.json({});

   } else {

res.json({

       email: req.user.email,

       id: req.user.id

     });

   }

 });

};

If the user is not logged in then an empty object will be sent. Or else the user’s email and id will be sent back. Sending a password is not a good idea.

16.  Html-routes.js

var path = require("path");

Path to use in relation routes for HTML files.

var isAuthenticated = require("../config/middleware/isAuthenticated");

module.exports = function(app) {

 app.get("/", function(req, res) {

If a user has an account then they will be sent to the members page.

if (req.user) {

     res.redirect("/members");

   }

   res.sendFile(path.join(\_\_dirname, "../public/login.html"));

 });

If a user has an account then they will be sent to the members page.

app.get("/members", isAuthenticated, function(req, res) {

   res.sendFile(path.join(\_\_dirname, "../public/members.html"));

 });

If an user who is not logged in tries to enter this area then they will be directed to the signup page.

17.  Package.json holds all packages installed.

18.  Server.js creates a server based application, example: localhost.

ar express = require("express");

var session = require("express-session");

Npm packages

var passport = require("./config/passport");

Passport as it has been formed.

var PORT = process.env.PORT || 8080;

var db = require("./models");

The setup of port and models needed for syncing.

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());

Showing apps needed for authentication.

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

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

Showing necessary routes.

db.sequelize.sync().then(function() {

 app.listen(PORT, function() {

   console.log("==> 🌎  Listening on port %s. Visit http://localhost:%s/ in your browser.", PORT, PORT);

 });

});

Syncing database and registering a message to the user who has had success using this app.

19.The  README.md explains the assignment.