WEB APP DEVELOPMENT

Assignment 2 Documentation

Web App Development

Lecturer: D. Drohan

The Internet of Things Yr. 3

Daniel Collins

Student No. 20076240

Contents

Introduction

The first Assignment required for this module involved creating a Node.js "back-end" sever application based on application developed for keeping track of players, teams and available pitches. Three main models were created, "Players", "Teams", and "Pitches". As per the requirements of the assignment, CRUD functionality was implemented on these models, giving the user the ability to Create, Read, Update and Delete objects within each model.
This server was linked to MLab which enabled the persistence of the data. It was deployed on Heroku, making it “live”, which enabled anyone to access the server and routes. As we did not have a “Client” side of the application, testing was done with the use of a RESTfulAPI HTTP Web Server Client. The application “Postman” was used for this purpose. When all the core routes were working, some smaller, more cosmetic routes were added in.

For the second assignment in this Module, a Client side of the application is required to be linked to the server application produced for Assignment 1. This will provide a User-Friendly Interface for running the routes and provide some additional functionality such as presenting the data in an aesthetic manner. The application shall be written using Vue.js and Webstorm, a programme from Intellij. This application provides the user with multiple different “html” pages by utilising “components”. These are Vue.js elements which can hold the html code, css styling and javascript methods in the same file, easing the difficulty of having a large application with methods and stylesheets having to be referenced from separate pages. The following document will discuss some of the main functionalities of this application and shall provide an overview of how they were achieved.

Assignment 1 – Recap of Functionality

The back end of an application is where most of the work is done. This will hold all the routes, the models for the components, and the practical and logical methods which eventually are referenced by the client server. This assignment contained four models from which the data objects were constructed. The four models used were;

- Pitches
- Players
- Teams
- Users

The schemas these models are based on are displayed below

Players Schema

```
let mongoose = require('mongoose');

let PlayersSchema = new mongoose.Schema({
      _id: {type: Number, default: 0},
      playerName: String,
      playerPosition: String,
      playerSport: String,
      playerAge: {type: Number, default: 1}
   },
   {collection: 'playerdb'});

module.exports = mongoose.model('players', PlayersSchema);
```

Pitches Schema

```
let mongoose = require('mongoose');

let PitchesSchema = new mongoose.Schema({
      _id: {type: Number, default: 0},
      pitchLocation: String,
      pitchLights: String,
      pitchSport: String,
      pitchAge: {type: Number, default: 1}
   },
   {collection: 'pitchdb'});

module.exports = mongoose.model('pitches', PitchesSchema);
```

Teams Schema

```
let mongoose = require('mongoose');

let TeamsSchema = new mongoose.Schema({
      _id: {type: Number, default: 0},
      teamName: String,
      teamLeague: {type: Number, default: 1},
      teamSport: String,
      numberOfPitches: {type: Number, default: 1}
   },
   {collection: 'rugbydb'});

module.exports = mongoose.model('teams', TeamsSchema);
```

Users Schema

```
let mongoose = require('mongoose');

let UsersSchema = new mongoose.Schema({
      _id: {type: Number, default: 0},
      email: {type: String, required: true, unique: true, match: /[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-
9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?/},
      password: {type: String, required: true}
   },
   {collection: 'usersdb'});

module.exports = mongoose.model('users', UsersSchema);
```

These schemas determine the rules an object will follow when being created. They list the required fields and their corresponding data types. As this application was linked to mLab, and the data was persisted, the models also contain the code for creating the collections within the database.

Routes

The routes are the connecting elements of the project. They specify what happens when an address is reached. Each model has a corresponding route object which hold the methods for each model. To delete a Player, for example, the delete function stored within the routes/players.js file would be called by the actual route element in the app.js file. The route in this file would specify the address required for the method to take place, followed by calling the method itself. If the address is incorrect or not reached, the method will not occur. A route in the app.js file follows the following template;

app.get('/users', users.findAll)

This specific route calls a "get" HTTP method, which is used when retrieving and displaying data. The path specified is "/users", so should that path not be reached, the method would not occur. The final section of the statement is telling the application what to do when it reaches this address. In this case, the application goes to the routes/users.js file, looks for the "findAll()" method and executes it. The application will then stay on this address until specified otherwise.
I have created the following routes, who's actions are self-explanatory;

Get's – Retrieving information

- app.get('/users', users.findAll);
- app.get('/teams', teams.findAll);
- app.get('/teams/totalTeams', teams.totalTeams);
- app.get('/players', players.findAll);
- app.get('/players/totalPlayers', players.totalPlayers);
- app.get('/pitches', pitches.findAll);
- app.get('/pitches/totalPitches', pitches.totalPitches);
- app.get('/users/totalUsers', users.totalUsers);
- app.get('/teams/:id', teams.findOne);
- app.get('/players/:id', players.findOne);
- app.get('/pitches/:id', pitches.findOne);
- app.get('/users/:email', users.findByEmail);
- app.get('/teams/sport/:teamSport', teams.findBySport);
- app.get('/players/playerPosition/:playerPosition', players.findByPosition);
- app.get('/pitches/pitchSport/:pitchSport', pitches.findBySport);
- app.get('/teams/teamLeague/:teamLeague', teams.findByLeague);
- app.get('/players/playerSport/:playerSport', players.findBySport);
- app.get('/pitches/pitchLocation/:pitchLocation', pitches.findByLocation);

Post's – Sending Information

- app.post('/addTeams', teams.addTeam);
- app.post('/addPlayers', players.addPlayer);
- app.post('/addPitch',pitches.addPitch);
- app.post('/addUsers/signUp',users.signUp);
- app.post('/users/login', users.login);

Put's – Updating Information

- app.put('/teams/:id/updateTeam', teams.updateTeam);
- app.put('/players/:id/updatePlayer',players.updatePlayer);
- app.put('/pitches/:id/updatePitch',pitches.updatePitch);

Delete's – Removing Information

- app.delete('/deleteTeam/:id',teams.deleteTeam);
- app.delete('/deletePlayer/:id',players.deletePlayer);
- app.delete('/deletePitch/:id',pitches.deletePitch);
- app.delete('/users/:id',users.deleteUser);

Assignment 2

Assignment 2 focus's on the client side of the application, that is, the user interface section. A Vue.js project was created for this and Vue 'components' were used to hold the code for the different pages. An 'index.html' file was created which holds the parent html code and inserts each of the components specific html code when needed. Each "path" was linked to a separate Vue.js component, which holds the code for all of the relevant methods and data for that page.
The project is deployed on the Firebase platform and

Functionality

The functionality of the client-side of this application is as follows

Login Page

Upon launching, the user is brought to the "login" page where they are asked to provide an email and password to log into an existing account. If the provided email and password do not match that of an account or they are not provided, an alert box notifies the user. At this point, with no user logged in, they are unable to access any of the pages of the application except the "login" and the "signUp" page.

SignUp Page

The "signUp" page gets the user to create an account using their first name, last name, email and password. Once created, an alert window informs the user the account has been created and automatically logs them in, bringing them to the home page of the application. At this point, the user has full access to the website.

Home Page

From the home page, there are links to "Manage", "Add" and "Members" dropdown sections, along with a "Map" section. The "Manage" dropdown holds the following options, "Manage Teams", "Manage Players", "Manage Pitches", the "Add" dropdown holds "Add Teams", "Add Players", "Add Pitches", while the "Members" dropdown holds "Log In", "Sign Out", and "Sign Up". As the user is already logged in, they are not able to access the "Log In" or "Sign Up" pages.

Manage Pages

The "Manage" pages are all the same, apart from the displayed content in the tables. The "Manage Players" page, for example, pulls the data from the MLab collection linked with "players" and displays this on a vue-table. Here, the user is able to see all of the significant data associated with each player, along with the options to filer the tables data. Each table row also includes two options, "Delete Player" and "Edit Player". Both the "Manage Teams" and "Manage Pitches" pages replicate this "Manage Players" page with their own relevant data.

Delete Pages

The user can delete a player by selecting the "Delete Player" button, represented by a Trash icon. A prompt will display asking the user if they are sure they would like to continue with the deletion process, giving them two options. The first, "Ok, Delete", continues with the delete and removes the selected item from the database, and then refreshes the table, proving this delete worked. Another prompt notifies the user of this. The second option on the first prompt, "Cancel" simply cancels the deletion request and returns to the "Manage Players" page.

Edit Pages

The user is also able to "Edit" a player from this table, by clicking on the "Edit Player" button, represented by an Edit icon. This will bring the user to an "Edit Player" page, which will display the current data as placeholders for the required fields. Once the form is filled out correctly, clicking on the "Save Changes" button at the end will save these edits to the player and return the user to the "Manage Players" page, showing the edits made on the table. If the form is not correctly populated, the user will be notified, and the page will not save the changes until the form is correct.

Add Pages

The "Add" pages, similar to the "Manage" pages, are all similar in their display, relevant to the specific path. The "Add Player" page displays a form for the user to complete. If a field is not correctly populated, or the form is incorrectly filled in, the user will be notified, and the page will not add the player until the form is sufficient. When the form is acceptable, the "Add Player" button adds this player to the database and notifies the user that the player has been added. The other "Add" pages follow this procedure.

Map Page

The "Map" page uses a Google Maps APi, displaying the users current location, if allowed by the browser (defaults to Waterford if not), on a GoogleMaps section. The search field at the top of the page provides the ability for the user to search for a location. Once the location is selected, the "Add" button places a marked at this location. Upon pressing the "Add" button, a prompt asks the user if they would like to add a pitch at this location. If the user selects yes, they are brought to the "Add Pitch" page. If they select no, the prompt disappears, showing the marker placed on the map at the specified location.

Member Pages

The "Members" dropdown contains three pages. If a user is logged in to the site, then two of these, "Log In" and "Sign Up", are inaccessible. The "Log Out" page signs the current user out of the site, notifying the member of this, and returns to the "Sign In" page. When no user is logged in, the "Sign In" and "Log In" pages are the only accessible pages on the site. The functionality of these pages has been described above.

Known Problems/Bugs Encountered

- The application is not responsive, therefor it will not display correctly on all browsers/screen sizes. For the application to display correctly on my browser, I have set the zoom level of chrome to 80%, which optimizes the display of the application, the navigation bar and the background image. Other zoom levels/screen sizes/operating systems/browsers may not display the application to its full potential.

- "Add" forms do not reset after successfully adding an object

- The first time the application is launched, the tables require an extended period of time to become populated with the data, presumably while the applications connects to MLab and retrieves the data from the appropriate database.

- The markers on the Map page are lost on each refresh/visit. If the "Map" page is left and returned to, all of the markers will have been removed.