Assignment 2 – SPA Blogging Website – Readme/Style Guide

By Joel Pillar-Rogers (word count: 950) Completed: 5 November 2019

1. Introduction

This document describes the structures, styles and functionality of my single page application. Key features include:

- MongoDB integration for storing News items
- SQLite integration for storing User details
- Login functionality that changes the content displayed
- Cookies to store session status

2. Project structure

The layout of my SPA is generally the same as in the Practicals. The root folder contains my "index.html" which is served to a visitor when they first arrive at the website (Figure 1). This folder also contains "server.js", which delivers the website and handles AJAX and CRUD operations. The json packages store NodeJS details and "Readme.md" is for my git repository. There are six folders in root that contain the rest of my content.

My index page uses ng-include and ng-view to display html stubs, which are kept in the View folder (Figure 2). Three stubs are always displayed: "title.html", "navbar.html" and "footer.html". The remaining stubs are only displayed when called by Angular's $routeProvider using similarly named deep linking routes (/, /create, /about, /faq and /login).

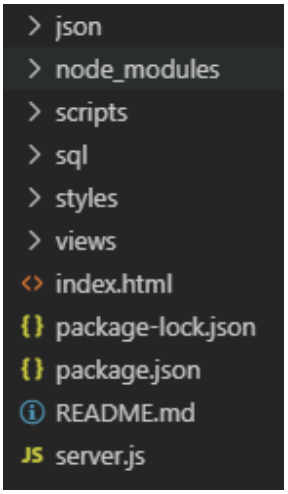

Figure 1. Root

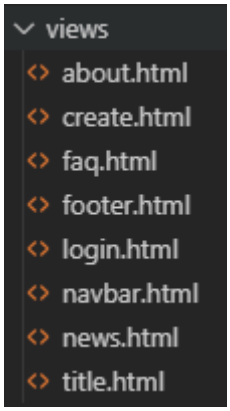

Figure 2. Views

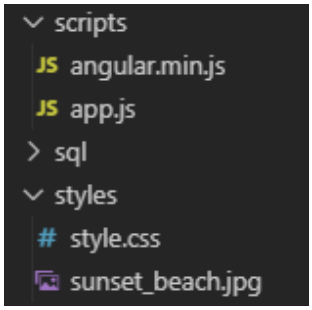

Figure 3. Scripts & CSS

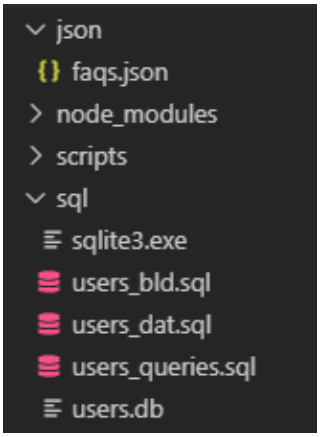

Figure 4. SQL & JSON

The SPA styling is provided by "styles.css" in the Styles folder and I changed the background image to "sunset_beach.jpg" (Figure 3). I continue to import "angular.min.js" to provide Angular functionality, and all of my front-end JavaScript is contained in "app.js" in the Scripts folder.

Finally, I have three stores of data used by the SPA, of which two are hosted directly on the server. I load the FAQ items from "faqs.json" in the JSON folder (Figure 4). I store user details in the SQLite database "users.db" in the SQL folder (this folder also has some files for editing the database directly, which won't be accessed during normal operation of the website). News items are stored in a remote MongoDB Atlas database.

3. Style guidelines

The style of my SPA is largely the same as the Practicals, besides some colour changes (I chose to focus on backend functionality for my extensions instead). The small changes I made include using "coral" instead of "dodgerblue" for all the highlighting and switching the background image to match the colour scheme. The font and layout are the same as the Practicals.

When a user logs into the website, several changes will occur:

- an "X" will appear next to each News item which will delete the item when clicked
- the Create form will appear on that page (unavailable otherwise)
- the "Login" button on the NavBar will change to "Profile"
- the login form will disappear and the user's details will be displayed instead

4. SPA Functionality

I added a lot of backend functionality, including databases, logins and cookies. I used RESTful AJAX calls in an AngularJS environment on the front-end. I used a NodeJS sever in an ExpressJS environment on the back-end. I used CRUD operations to interact with my databases.

When the server is started, it will initialise connections with the remote MongoDB Atlas database and with the local SQLite database. To establish these connections, I installed the Node packages "mongodb" and "sqlite3" and adapted the instructions on these websites:

- https://expressjs.com/en/guide/database-integration.html
- https://www.thepolyglotdeveloper.com/2018/09/developing-restful-api-nodejs-mongodb-atlas/
- https://www.sqlitetutorial.net/sqlite-nodejs/connect/

News items are stored in MongoDB Atlas and in a temporary storage structure (an array) in "app.js". When the website is first launched, a GET request to retrieve news items is sent to the server, which uses a READ request to access the items in the MongoDB database and return them. These items are stored in the temporary array. When an item is created, it is stored in the temporary array and a POST request is sent to the server containing the data. The server adds this data to the MongoDB database using a CREATE request on the open connection. Similarly, a DELETE request is sent whenever the "X" is clicked on a News item (after logging in), and this is also handled by the server. There is no UPDATE functionality.

Note: The login details for my MongoDB database are: Username: pill0032@flinders.edu.au Password: _,L#M3sZPJQRrxL It is a brand-new account, and this is the only database on it, so feel free to open it and explore. I will change the password next semester.

To populate the **FAQ page**, another GET request is sent to the server when the webpage loads. The server accesses the local "faqs.json" file to retrieve the requested items and return them. These FAQ items are also stored in a temporary array.

When a user **logs in** to my SPA with a username and password, these details are sent via a GET request to the server, which queries them against stored users in the local SQLite database. There is no validation of user input and passwords are stored unsalted. If it finds a matching username and password, it will return a successful response along with that user's data. This returned data will be displayed on a Profile page that will now appear. The user's status is set to "logged in" using a $rootScope variable.

These are the stored login details you can use to test the website functionality:	
Username: user1	Password: pass1
Username: user2	Password: pass2
Username: user3	Password: pass3
Username: user4	Password: pass4

When a user logs in successfully, a **cookie** is created and stored on the user's machine. This saves the "logged in" status, as well as the user's personal details. If the page is reloaded, this cookie is checked to see whether the user is still logged in and the appropriate content is displayed if they are. The cookie lasts for 30 days. Once logged in, on the Profile page there is a Logout button which will return the website to a logged-out status and delete the stored cookie.

The server is launched by typing "node server.js" at the command line; the port used is 3000.