Project Guide

***The 3 Amigos***

***Wolfgang Brevik & Erin Crump***

***Bucket List Tours Website***

1. **Application URL:**

<https://bucket-list-tours.herokuapp.com/>

1. **Application Summary:**

The purpose of this application is to allow the user to save, review, and purchase different bucket list tours. The user is directed to a home page that displays all tours available and can click on the individual tour which displays information about that tour. On this individual tour page, the user can read reviews, view the tour guides, interact with a map api that pinpoints the different stops on the tour. The user can also sign up for tours by creating an account and can book/buy a tour.

Additionally, the user can create an account using the “Sign up” button in the right-hand corner of the navigation bar. Once the account is created, the user can click on their name (where the login/ sign up buttons were) that will redirect them to their user setting page where they can update their account credentials and even upload a profile picture. Additionally, there are links that allow the user to look at their booked tours, check their billing information, and look at their past reviews.

The admin has some unique privileges when they log-in. The admin can perform all the functions that a normal user can, but they are able to also manage tours (all CRUD functions), manage users accounts, manage reviews, and manage bookings.

1. **Routing Guide:**

GET / : Displays the home page that shows a panel of all tours from the database.

GET /login : Displays the login page and allows users to enter credentials.

POST /login : Signs user into account through passport and establishes a JWT session and provides a bearer tag.

GET /signup: Displays the sign up page that allows users to create an account.

POST/signup: Creates a user in the database and redirects to the homepage.

GET /settings/:id : displays the users account settings and links to various other pages associated with the user.

POST /submit-user-info: Allows the user to enter a new username/ email or password and automatically updates the information in the database.

GET /manage-users : Displays all users in the database.

GET /tour/:id : Displays individual information for a tour.

**Routes We Couldn’t Get To (Still returns JSON Data):**

GET /manage-users/:id : Displays a view where the admin can look at an accounts settings and update or delete the account.

POST /manage-users/:id/update : Updates or deletes users account

GET /manage-tours/ : Displays all tours in the database with the option to update or delete each, or create a tour.

GET /manage-tours/new: Displays a form to create a new tour and upload a photo for that tour and assign tour guides.

POST /manage-tours/create: Creates a new tour instance in the database.

GET /manage-tours/:id : Displays a view where the admin can update or delete the a tour.

POST /manage-tours/:id/update: Updates or deletes a tour instance in the database.

**API Routes:**

**POSTMAN**

**All Tours: GET** <https://bucket-list-tours.herokuapp.com/app/tours>

**LogIn:** **POST** <https://bucket-list-tours.herokuapp.com/app/users/login>

* Set body to raw -> JSON
* {“email”: [admin@user.com](mailto:admin@user.com), “password”: “pass1234”}

**TopThree: GET** <https://bucket-list-tours.herokuapp.com/app/tours/top-three>

**CheapFive: GET** <https://bucket-list-tours.herokuapp.com/app/tours/cheap-five>

**I will provide and export of all the other routes**

1. **User Credentials and Roles:**

* Administrators:
  + Exclusive Route Access: /manage-users, /manage-users/:id
  + Sample User Account: username: [admin@user.com](mailto:admin@user.com) password: pass1234
* User:
  + Exclusive Route Access: /settings
  + Sample User Account: username: [erin@test.com](mailto:erin@test.com) password: 12345678

1. **Bug Hunt Response:**

|  |  |
| --- | --- |
| **Comment** | **Response** |
| Log out button wasn’t functioning correctly | Fixed it. Pushed the project again to Heroku (worked locally, just not on the Heroku site). |
| Image upload on user settings wasn’t working | Fixed it. Used a different package called “multer” instead of the one used in the Mustacchio project. |
| Some links on the user settings page and admin settings page didn’t return a view. | Wasn’t able to fix all of these in the time left. We left them as they still return JSON data that proves we returned multiple things from our database. |
| (He didn’t write anything on the document provided, this is just the comments we remember from our discussion in class.) | We fixed the items he mentioned in the discussion. |

1. **Project Requirements**

|  |  |
| --- | --- |
| 1. 5 interlinked dynamic pages (views) that display dynamic, database-driven content per team member. To be counted as dynamic, the page must retrieve information to and/or write information to the application database using one or more queries/data access functions written by the team.  2. At least 5 database transactions (a transaction includes selecting, inserting, updating, or deleting data from the database) per team membe**r** | Views that display dynamic content from database and associated queries/transactions:   1. All Tours    1. lists all tours and       1. starting point       2. next start date       3. total per group       4. how many stops       5. price       6. rating average 2. Log In    1. validates user and provides a token 3. Sign Up    1. creates an account for the user 4. Settings    1. User Account Settings       1. Name       2. email address       3. user image 5. Update User Settings (image,name,email) 6. Top three tours    1. starting point    2. name    3. summary    4. next start date    5. total per group    6. how many stops    7. price    8. rating average 7. Top Five cheap tours    1. starting point    2. next start date    3. name    4. summary    5. total per group    6. how many stops    7. price    8. rating average 8. manage users    1. user image    2. user name    3. user email 9. Tour details    1. Tour guides    2. overview    3. about content    4. images    5. starting point    6. duration    7. reviews 10. Map     1. pulls coordinates from the database and plots points |
| 3. Support for user authentication and secure storage of user credentials | For this requirement, we mimicked what we did in class with Mongoose (this authentication is all done through the authenticationContorller.js file and is well documented). We also used a JSON web token for authentication. We were originally going to try to incorporate the Passport package, but it didn’t work very well with our template. |
| 4. Preservation of state (may include cookies, session state, url parameters, querystring, etc.) | We used cookies and JSON web tokens to create a preservation of state when you log into your account. |
| 5. Development of at least one REST API and consumption of another third-party API | We used an API called Mapbox which allows us to show the stop data from the database for each tour on the individual tour views. This API also allows the user to scroll around to look at the map in all directions including zooming in. They also let us customize the view of the map so that it has an aesthetic that goes well with our website design. |
| 6. Inclusion and use of at least two npm libraries that are not covered elsewhere in the course (i.e., you need to explore additional functionality on your own). | We used multer to upload admin/user photos  We used pug for the views  We used slugify  We used bcryptjs for the encryption of the web token  We also used helmet, morgan, cookieparser, eslint (error handling), and dotenv |
| 7. Input validation and error handling for all free-form user entries that could potentially result in errors | Input Validation: The sign up and login pages have error messages that pop up if certain parameters are not met (minimum character requirements for passwords, valid email format). The settings page as input validation for creating a new password or uploading certain files (must be jpeg or png, if its a csv it won’t take the upload).  Error Handling: Some of the routes we didn’t have time to get to return just the JSON error and messaging instead of an error view. |