PROJECT OVERVIEW

(Last Updated: 8/13/2024)

**Pre Planning**

Website objectives

| **Objective** | **Completed** |
| --- | --- |
| Set-up | yes |
| Skeleton | yes |
| Post Back End | no |

### User Can Access and View a Basic White Webpage

**As a** new visitor,  
**I want to** be able to visit a simple, bare-bones white website,  
**So that** I can verify that the basic hosting setup is correct and the page is accessible online.

### Acceptance Criteria:

1. **Basic Webpage Setup:**
   * The webpage is hosted and accessible through a standard web browser.
   * The webpage displays a plain white background with minimal content.
2. **Page Accessibility:**
   * The webpage loads successfully without errors when accessed via its URL.
   * The load time is minimal.

### Task Completion Criteria:

**Initial Setup:**

* Install Node.js and npm on the machine.
* Initialize npm and ensure it is running without issues.
* Generate the package.json file.

**Dependencies:**

* Install Express.js using npm install express.
* Install Handlebars using npm install express-handlebars.
* Install Tailwind CSS.
* Install pg (PostgreSQL client).

**Server Configuration:**

* Set up server.js.
* Configure Handlebars in the server.
* Create a basic Handlebars template and a homepage.

**Finalizing Setup:**

* Configure Express to serve static files from the public folder.
* Configure Tailwind CSS.
* Ensure everything is working by running node server.js.
* Open a web browser and navigate to http://localhost:3000 to check that all functions as expected.

**Deployment:**

* Set up a new project on Render and connect it to the GitHub repository.
* Configure deployment settings according to the Node.js environment.
* Launch the website online and verify its functionality.

### User Can Access and View a Basic White Webpage

**As a** new visitor,  
**I want to** be able to visit a simple, bare-bones white website,  
**So that** I can verify that the basic hosting setup is correct and the page is accessible online.

### Acceptance Criteria:

1. **Basic Webpage Setup:**
   * The webpage is hosted and accessible through a standard web browser.
   * The webpage displays a plain white background with minimal content.
2. **Page Accessibility:**
   * The webpage loads successfully without errors when accessed via its URL.
   * The load time is minimal.

### Task Completion Criteria:

**Initial Setup:**

* Install Node.js and npm on the machine.
* Initialize npm and ensure it is running without issues.
* Generate the package.json file.

**Dependencies:**

* Install Express.js using npm install express.
* Install Handlebars using npm install express-handlebars.
* Install Tailwind CSS.
* Install pg (PostgreSQL client).

**Server Configuration:**

* Set up server.js.
* Configure Handlebars in the server.
* Create a basic Handlebars template and a homepage.

**Finalizing Setup:**

* Configure Express to serve static files from the public folder.
* Configure Tailwind CSS.
* Ensure everything is working by running node server.js.
* Open a web browser and navigate to http://localhost:3000 to check that all functions as expected.

**Deployment:**

* Set up a new project on Render and connect it to the GitHub repository.
* Configure deployment settings according to the Node.js environment.
* Launch the website online and verify its functionality.

### User Can See a Rough Unstyled Outline of the Website

**As a** new visitor,  
**I want to** be able to see a simple unstyled website,  
**So that** I can verify the website is well-structured and functional.

### Acceptance Criteria

**Basic Website Layout:**

* The website displays the basic structure of the blog.
* The website can be navigated using the navbar.

**Page Accessibility:**

* The webpage loads successfully without errors when accessed via its URL.
* The load time is minimal.

### Task Completion Criteria

**Set-up:**

* Change the HTML <title> to “Joshua’s Corner.”
* Add necessary meta tags for SEO and responsiveness.

**Pages:**

* Create the Dashboard page and its route.
* Create the Log page and its route.
* Test that each url links to the proper page.

**Root Structure:**

* Establish a root structure in the main Handlebars template.
  + Add a header section inside the root.
  + Add a body section inside the root.

**Header:**

* Implement a header that is visible across all pages.
* Set the header title to “Joshua’s Corner.”
* Ensure the header has a minimum width and includes a border.

**Navbar:**

* Construct an unstyled Navbar that is visible across all pages.
* Include links to the Home, Dashboard, and Log pages.
* Ensure clicking the links redirects to the corresponding pages.

**Body:**

* Design a body section with a minimum height and a border.

**Blog Entry:**

* Create a mockup blog entry.
  + Include a title.
  + Display “posted by: User” on [DATE] line with today's real date.
  + Include a body paragraph of Lorem Ipsum.

### User Can See All the Posts in the Database

**As a** new visitor,  
**I want to** be able to see all the posts submitted to the site,  
**So that** I can verify if the database is properly set up for blog entries.

### Acceptance Criteria

**Basic Functioning Back-End:**

* The homepage displays all the posts stored in the database.
* Each blog post displays the username of the author, the date of creation, and the content.

### Task Completion Criteria

**Set-Up:**

* Install PostgreSQL on the server.
* Create a database for storing blog posts.
* Install dotenv
* Add a .env file with database credentials to ensure secure access.
* Establish communication between the site and the database.

**Post Model:**

* Create a Post model in the database with the following attributes:
  + id: Unique identifier for the post.
  + title: Title of the post, displayed at the top.
  + author: Username of the person who created the post.
  + date\_of\_creation: Timestamp for when the post was created.
  + content: Main text content of the post.

**Post JSON:**

* Develop a function to convert raw database entries into a consistent JSON format. This function should also validate the presence of all necessary information in each post.

**Post Creation:**

* Implement a factory function that accepts title, author, and content as parameters and creates a new post instance in the database.

**Get Post:**

* Create a function that retrieves a specific post by its id and returns it as a JSON object.

**Update Post:**

* Develop a function that updates an existing post based on a modified JSON object that includes the post's id.

**Get All Posts:**

* Implement a function to retrieve all posts from the database and return them as a list of JSON objects.

**Delete Post:**

* Create a function that deletes a post from the database given its id.

**Test Post:**

* Develop a test suite to ensure the translator correctly formats data for the website and its author.
* Develop a test suite to ensure posts will successfully success and fail when expected.
* Develop a test suite to ensure post can be retrieved from database successfully.
* Develop a test suite to ensure post can be updated from database successfully.
* Develop a test suite to ensure all the post can be retrieved when updatePost is called.
* Develop a test suite to ensure post will be successfully deleted from the database upon calling deletePost function.

### User Can Sign In, Log In, and Log Out

**As a new visitor,**

I want to be able to sign in, log in, and log out of the website so that I can manage my account securely and access personalized features.

**Acceptance Criteria:**

* Users can sign in
* Users can log in
* Users can log out
* Passwords are protected
* Sessions are implemented

**Task Completion Criteria:**

**Setting Up the Suite:**

* Create a User model file
* Create a User route file

**User Sign In:**

* Users can create an account using tools like Postman or Insomnia.
* Users must provide a username and a password to sign in.
* The password will be encrypted upon creation.
* If the user cannot sign in, the system will return an error.

**User Log In:**

* Users can access their account by submitting their credentials through Postman or a similar tool.
* If the username or password is incorrect, the system will return an error.
* If the credentials are correct, the user will receive an object containing their account data.

**User Log Out:**

* After logging out, users will receive a message confirming that they have been logged out.
* If the user is not logged in, they will receive a message notifying them of this issue.

**Session Management:**

* A session will be created when the user logs in and destroyed when the user logs out.

**Reliability:**

* Test the sign-in functionality
* Test the log-in functionality
* Test the log-out functionality

### User Can Sign In, Log In, and Log Out On Website

As a new visitor to the blog,  
I want to sign up, log in, and log out effortlessly,  
So that I can post say things online.

### Acceptance Criteria:

* User can see a sign-up and log-in page when visiting the authentication endpoint.
* User can create a new account by filling out the sign-up form.
* User can log in using the credentials created during sign-up.
* User can log out of their account by clicking the "Logout" button.
* User can only see the "Logout" button if they are logged in.

**Task-Completion Criteria:**

**Sign Up:**

Users can sign up using a form on the authentication page.

The form will include fields for:

Username

Password

Upon submission, the form will make a POST request to the endpoint api/users/.

* The sign-up process must be tested to handle the creation of up to 7 unique users without issues.
* Validation must include the analysis but subsequent dismissal for duplicate usernames and ensuring appropriate error handling.

**Log In:**

* Users can log in using a form on the authentication page.
* The form will include fields for:

Username

Password

* Upon submission, the form will make a POST request to the endpoint api/users/login.
* The log-in process must be tested to handle 7 consecutive sign-ins and sign-outs without failures.

**Log Out:**

* A log-out button will be available, but only visible when the user is logged in.
* When clicked, the button will make a POST request to the api/users/logout endpoint.
* Upon successful logout, the button should disappear .
* Test the log-out process.