National College of Ireland

HDSDEV_SEP23

2023/2024

Alexandru

23124091

x23124091@student.ncirl.ie

SoulJournal

Technical Report

National College of Ireland

Project Submission Sheet – 2022/2023

Student Name: Alexandru Georgescu

Student ID: 23124091

Programme: Higher Diploma In Software Development (HDSDEV_SEP23)

Year: 1

Module: Project

Lecturer: Hamilton Niculescu

Submission Due Date: 10/08/2024

Project Title: **SoulJournal**

Word Count (excluding bibliography and appendices):

Signature:

Alexandru Georgescu

Date: 10/08/2024

Table of Contents

Table of Figures

A table containing all the figure number, description and page

Glossary, Acronyms, Abbreviations and Definitions

A table containing all the terms and acronyms used in the document.

Executive Summary

SoulJournal addresses the challenge of maintaining mental well-being through regular journaling. Many individuals struggle to keep a consistent journaling habit due to lack of motivation, organization, and accessibility(TheHappyJournals).

Souljournal is a web-based application designed to simplify and enhance the journaling experience by providing an intuitive and user-friendly platform.

The technical solution involves a React-based frontend integrated with a backend server that manages user authentication, journal entries, and feedback. The application leverages modern web technologies such as React Router for seamless navigation, Bootstrap for responsive design, and RESTful APIs for efficient data handling. Key features include user registration and login, creating and managing journal entries, receiving motivational quotes, and providing feedback.

The evaluation of **SoulJournal** was conducted through user testing and feedback collection. Users described that the web application is designed in such a way that you’re not distracted by images, useless text, and you’re keeping your focus on the new entry.

Overall, **SoulJournal** successfully provides a supportive environment for users to maintain their journaling habits, contributing positively to their mental well-being.

1 Introduction

1.1 Background

“Research” by Laura King shows that writing about achieving future goals and dreams can make people happier and healthier. Similarly, Jane Dutton and Gregory Ciotti found that when people doing stressful fundraising jobs kept a journal for a few days about how their work made a difference, they increased their hourly effort by 29% over the next two weeks.” (Source)

A web-based journaling application also offers unparalleled convenience, allowing users to journal from any device with itnernet access. This flexibility is particularly appealing in today’s fast-paced world, where individuals seek tools that fit seamlessly into their busy lifes.

1.2 Aims

The objective for this project is to build a journaling web application called **SoulJournal**. The web application will be designed to provide users a convenient and secure platform for personal journaling. **SoulJournal** will seek to address common barriers such as lack of motivation, organization, and accessibility.

SoulJournal aims to:

1. Promote Consistent Journaling: By offering an easy-to-use interface and motivational features, Souljournal encourages users to maintain a regular journaling habit.
2. Enhance user Experience: Utilizing modern web technologies, the application provides a seamless and engaging user experience, ensuring that users can easily navigate and utilize the platform.

3. Support Mental Health: By facilitating regular journaling, **SoulJournal** aims to help users manage stress, regulate emotions, and improve self-awareness, contributing to overal mental well-being.
4. Ensure Accessibility: **SoulJournal** is designed to be accessibile on various devices, ensuring that users can journal anytime and anywhere.

1.3 Technologies

Technologies used in the **SoulJournal** Project:

1. **React**
    - **Description**: React is a JavaScript library for building user interfaces, particularly single-page applications where you need a fast, interactive user experience.
    - **Contribution**: React is used to build the front-end of the **SoulJournal** application. It allows for the creation of reusable components, efficient state management, and dynamic rendering of the user interface. This makes the application responsive and interactive.
2. **JavaScript (ES6+):**
    - **Description**: JavaScript is a programming language that enables interactive web pages. ES6+ refers to the latest versions of JavaScript, which include new syntax and features.
    - **Contribution**: JavaScript is the primary language used to write the logic for the application. ES6+ features like arrow functions, destructuring, and async/await make the code more concise and easier to manage.
3. **HTML5**
    - **Description**: HTML5 is the latest version of the HyperText Markup Language, which is used to structure content on the web.

- **Contribution**: HTML5 is used to structure the content of the **SoulJournal** application. It provides the basic elements and semantic tags that form the foundation of the web pages.

4. **CSS3**

- **Description**: CSS3 is the latest version of the Cascading Style Sheets language, used to style HTML elements and control the layout of the web pages.
- **Contribution**: CSS3 is used to style the **SoulJournal** application, making it visually appealing and user-friendly. It ensures that the layout is responsive and consistent across different devices.

5. **Bootstrap**

- **Description**: Bootstrap is a popular front-end framework for developing responsive and mobile-first websites.
- **Contribution**: Bootstrap is used to quickly design and customize responsive web pages. It provides pre-designed components and a grid system that helps in creating a consistent layout and styling across the application.

6. **React Router**

- **Description**: React Router is a library for routing in React applications. It allows for navigation between different components and views.
- **Contribution**: React Router is used to manage the navigation within the **SoulJournal** application. It enables users to move between different pages(“Home”, “Journal”, “Quotes”) without reloading the entire application.

7. **Node.js**

- **Description**: Node.js is a JavaScript runtime built on Chrome’s V8 JavaScript Engine. It allows for server-side scripting using JavaScript.

- **Contribution**: Node.js is used for the back-end of the **SoulJournal** application. It handles server-side logic, database interactions, and API requests, enabling a seamless connection between the front-end and back-end.

8. **Express.js**
   - **Description**: Express.js is a web application framework for Node.js, designed for building web applications and APIs.
   - **Contribution**: Express.js is used to create t he server and define the API endpoints for the **SoulJournal** application. It simplifies the process of handling HTTP requests and responses.

9. **MongoDB**
   - **Description**: **MongoDB** is a **NoSQL** database that stores data in JSON-like documents.
   - **Contribution**: **MongoDB** is used to store the data for the **SoulJournal** application, such as user entries, quotes, and other journal-related information. Its flexible schema allows for easy storage and retrieval of data.

10. **Mongoose**
    - **Description**: Mongoose is an Object Data Modeling(ODM) library for MongoDB and Node.js
    - **Contribution**: Mongoose is used to interact with the MongoDB data base. It provides a schema-based solution to model the application data, making it easier to validate and manage.

11. **Visual Studio Code**
    - **Description**: VS Code is a source-code editor developed by Microsoft. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, and more.

- **Contribution**: VS Code is the development environment used to write and manage the project's code. Its features like extensions and integrated terminal puts in order the development process.

12. **Git**

- **Description**: Git is a distributed version control system used to track changes in source code during software development.
- **Contribution**: Git is used for version control in the **SoulJournal** project. It allows multiple developers to collaboarte, track changes, and manage different versions of the codebase.

These technologies work together to create a full-stack web application that allows users to create, manage, and view journal entries.

1.4 Structure

1. **Introduction:**
   - **Background:** Provides context and motivation for the SoulJournal project.
   - **Aims:** Outlines the primary goals and objectives for the project.
   - **Technologies Used:** Lists the technologies and tools utilized in the development of SoulJournal.
   - **Structure:** Gives an overview of the documentation structure, detailing the contents of each section.
2. **System:**
   - **Requirements:**
     - **Functional requirements:** Specifies the functionalities that the system must provide.
     - **Data requirements:** Describes the data that the system will handle.
     - **User requirements:** Outlines the needs and expectations of the end-users.
     - **Environmental requirements:** Lists the environmental conditions under which the system will operate.
     - **Usability requirements:** Defines the usability criteria for the system.
   - **Design and Architecture:** Describes the overall architecture and design principles of the SoulJournal application.
   - **Implementation:** Details the implementation of key features and components.

- **Testing:** Covers the testing strategies and quality assurance practices.
- **Graphical User Interface(GUI) Layout:** Discusses the design principles and user interface elements.
- **Customer Testing:** Describes the process and results of testing the system with actual users.
- **Evaluation:** Summarizes the evaluation of the system against the requirements and objectives.

3. **Conclusions:** Summarizes the project outcomes, including achievements and challenges faced.
4. **Further development or research:** Discusses potential future enhancements and areas for further research.
5. **References:** Lists all the references and sources used in the documentation.
6. **Appendix:**
   - **Project Proposal:** Includes the intiail project proposal document.
   - **Project Plan:** Contains the detailed project plan.
   - **Requirement Specification:** Provides the complete requirement specification.
   - **Monthly Journal:** Includes the monthly journal entries documenting the project progress.
   - **Other Material Used:** Lists any additional materials used during the project.

2 System

2.1 Requirements

The following section outlines all the functional requirements that **SoulJournal** should accomplish and it also describes the main details of the systems and the way the users will interact with the system. We will also investigate non-functional requirements

2.1.1 Functional requirements

The functional requirement describes the core features and functionalities that the SoulJournal system must provide. The requirements are ranked in terms of priority (1 being the highest priority).

1. **User Registration and Authentication**
    - Non-registered users should be able to create a new account by providing necessary information (email, password).
    - Registered users should be able to securely log in to the web application.
2. **Journal Entry Management**
    - Users should be able to create new journal entries.
    - Users should be able to save journal entries securely.
    - Users should be able to retrieve and view their previous journal entries.
    - Users should be able to update or delete their journal entries.
3. **Quote Integration**
    - The application should integrate with the **Quotable** API to fetch and display inspirational quotes.
    - Users should be able to filter or search for quotes based on specific topics.

4. **User Interface and Customization**
   - The application should provide a user interface focused on the writing experience.
   - Users should be able to customize the application's appearance, such as themes, fonts and layouts.
5. **User Feedback and Suggestions**
   - Users should be able to submit feedback, suggestions, or report issues within the application.

1. **Top Level Use Case Diagram**

The following diagram outlines the actors and how they interact with the **SoulJournal** web application(Figure 1.1)

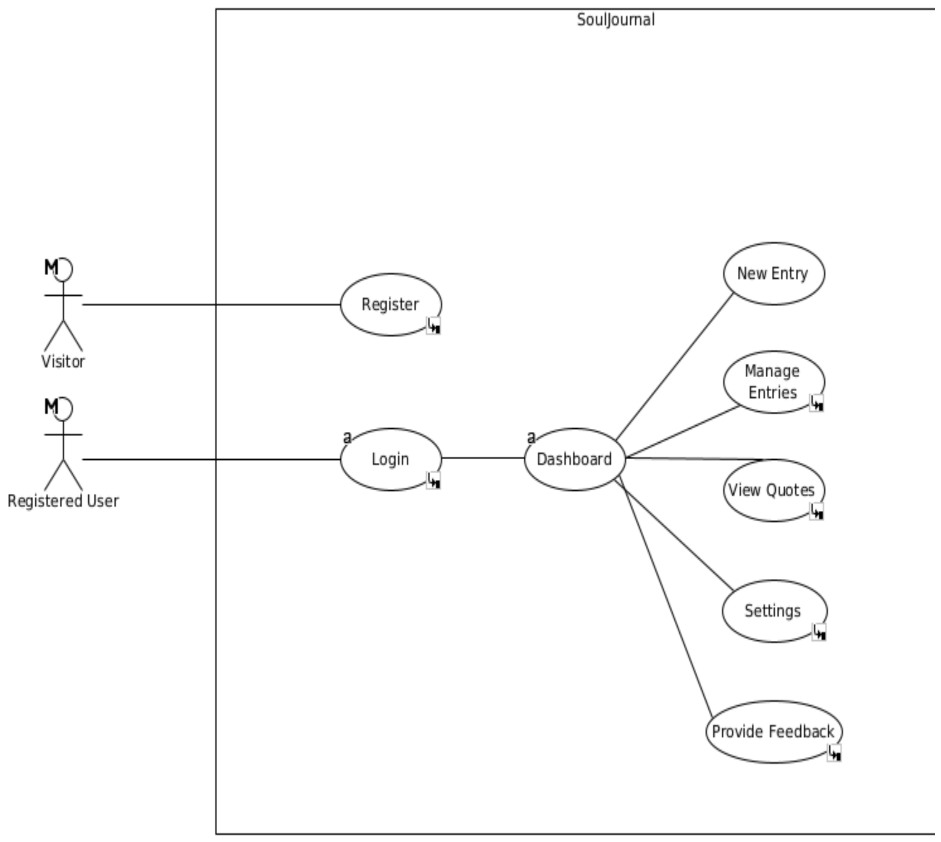


Figure 1.1: Top-Level Use-Case Diagram

2. Register

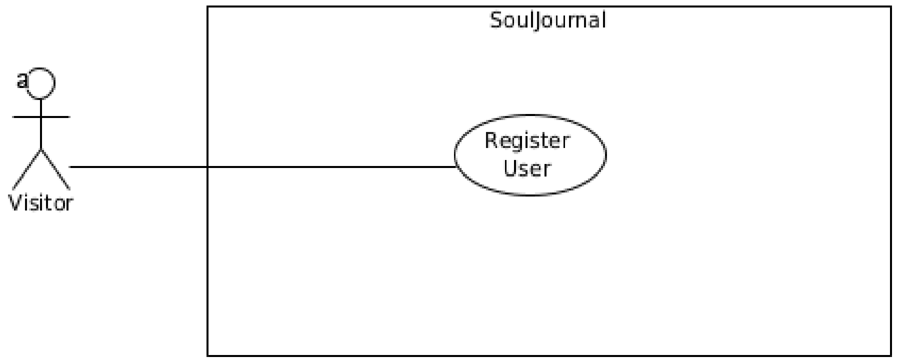


Figure 2.1: Register User

Name	**Register Customer**
Description	A non-registered user registers with the system
Priority	This is a **high** priority requirement.
Scope	This allows new users to create an account and access the full functionality of the **SoulJournal** application.
Precondition	The user is not registered with the system.
	Flow Description
Activation	This use case starts when an unregistered visitor attempts to create a new account.
Main Flow	• The user selects the option register.

- The system displays the registration form.
- The user enters their personal information(first name, last name, email, password).
- The user submits the registration form.
- The system validates the provided information.
- The system creates a new user account.
- The system displays the Login page

Alternate Flow

Title	Description
The user cancels the registration process	A1.1. The user cancels the registration process. A1.2. The use case terminates

Exceptional Flow

Title	Description
The provided information (email/password) is invalid	E1.1. The system displays an error message. (**Title**) E1.2. The user is prompted to correct the information. E1.3. The use case continues from main flow 3.

Post-Condition (For Successful Main Flow)

Title	Description
Termination	The system displays registration is successful and the user is given the option to go to the Login section in the SoulJournal application

3. Login

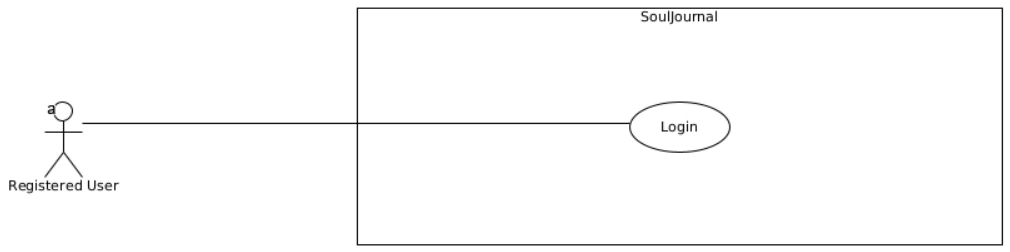


Figure 3.1: Login

Name	**Login User**
Description	A registered user logs in the web application
Priority	This is a **high** priority requirement.
Scope	This use case describes the process of a registered user authenticating themselves by providing their credentials to gain access to the SoulJournal application.
Precondition	The user has a registered account.
	Flow Description
Activation	This use case starts when a registered user attempts to log in to the application.
Main Flow	1. The system displays the login form. • The user enters their information (email,

password).

- The user submits the login form.
- The system validates the provided information.
- The system logs the user into their dashboard

Alternate Flow

Title	**Description**
N/A	N/A

Exceptional Flow

Title	**Description**
The provided information (email/password) is invalid	E1.1. The system displays an error message.(**Title**) E1.2. The user is prompted to enter the correct information. E1.3. The use case continues from main flow 2.

Post-Condition (For Successful Main Flow)

Title	**Description**
Termination	The user is logged in to the SoulJournal application dashboard.

4. New Entry

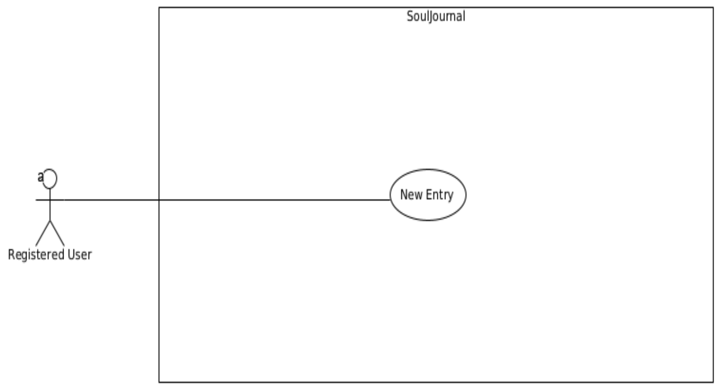


Figure 4.1.: New Entry

Name	**New Entry**
Main Flow	• The user selects the "New entry" action. • The system displays a new, blank journal entry form. • The user fills out the form and submits it. • The system saves the new journal entry. • The system updates the database to include the new entry.
	Alternate Flow
User Cancels Creation	A1.1. The user cancels the action. A1.2. The system discards any input. A1.3. The use case terminates.
	Exceptional Flow

Title	Description
Error Saving Entry	E1.1. The system encounters an error saving the entry. E1.2. The system displays an error message. (**Title**) E1.3. The user is prompted to try again or cancel. E1.4. If the user retries, the use case continues at step 3 of the main flow. E1.5. If the user cancels, the system discards any input and the use case terminates.

5. Journal Entries

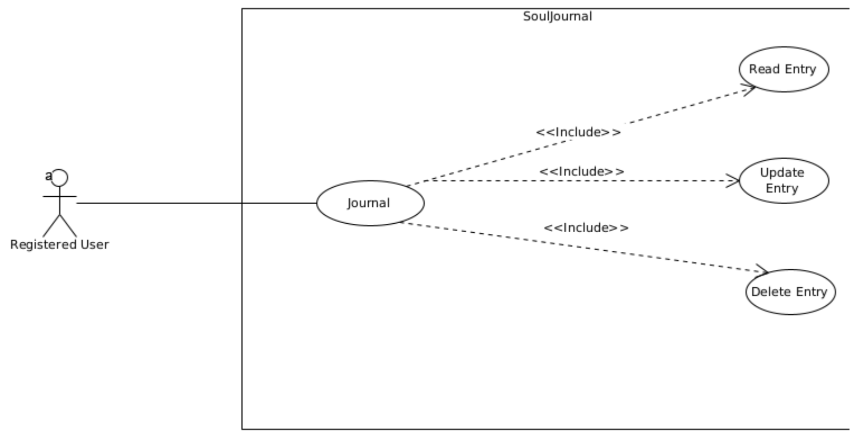


Figure 5.1: Journal Entries

Name	Manage Entries

Description A registered user can **R.U.D.** (Read, Update or Delete) their journal entries.

Priority This is a **high** priority requirement.

Scope This use case enables registered users to **R.U.D. (**Read, Update or Delete) their personal journal entries within the **SoulJournal** application.

Precondition The user is logged in to the SoulJournal application

Flow Description

Activation This use case starts when a logged-in user wants to manage their journal entries.

Main Flow

1. The system displays the user’s journal entries.
   - The user selects an action (read, update, or delete).
   - The system executes the selected action on the journal entry.
   - The system updates the display of journal entries.

Individual Flow Descriptions for Journal Entries

Read Entry

Name **Read Entry**

Main Flow

1. The user selects the “Read” action.

2. The system displays the selected journal entry in read-only mode.

Alternate Flow

User Exits Read Mode	A1.1. The user exits the read mode. A1.2. The use case terminates.

Exceptional Flow

Title	**Description**
Error Loading Entry	E1.1. The system encounters an error loading the entry. E1.2. The system displays an error message. **(Title)** E1.3. The user is prompted to try again or cancel. E1.4. If the user retries, the use case continues at step 2 of the main flow. E1.5. If the user cancels, the use case terminates.

Update Entry

Name	**Update Entry**
Main Flow	• The user selects the “Edit” action. • The system displays the selected journal entry in edit mode. • The user modifies the entry and submits the changes. • The system saves the updated entry. • The system updates the display to reflect the changes.

Alternate Flow

User Cancels Creation	A1.1. The user cancels the action. A1.2. The system discards any changes. A1.3. The use case terminates.

Exceptional Flow

Title	**Description**
Error Saving Entry	E1.1. The system encounters an error saving the updated entry. E1.2. The system displays an error message. **(Title)** E1.3. The user is prompted to try again or cancel. E1.4. If the user retries, the use case continues at step 3 of the main flow. E1.5. If the user cancels, the system discards any changes and the use case terminates.

Delete Entry

Name	**Delete Entry**
Main Flow	• The user selects the "Delete" action. • The system prompts the user to confirm the deletion. • The user confirms the deletion. • The system deletes the journal entry.

- The system updates the journal entries.

Alternate Flow

User Cancels Deletion	A1.1. The user cancels the action. A1.2. The use case terminates.

Exceptional Flow

Title	**Description**
Error Deleting Entry	E1.1. The system encounters an error deleting the entry. E1.2. The system displays an error message. **(Title)** E1.3. The user is prompted to try again or cancel. E1.4. If the user retries, the use case continues at step 3 of the main flow. E1.5. If the user cancels, the use case terminates.

6. Quotes

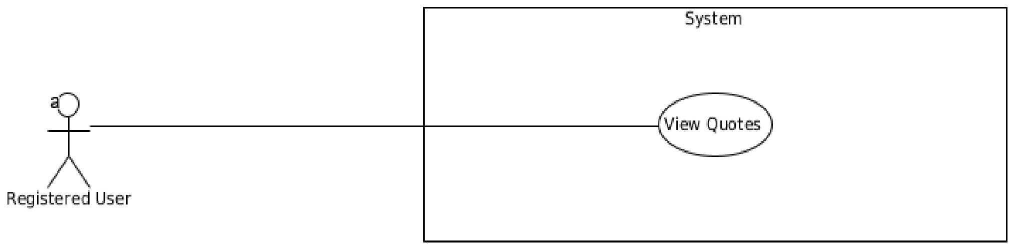


Figure 6.1: Quotes Use Case

Name	**Quotes**
Description	This allows registered users to view inspirational quotes fetched from the Quotable API.
Priority	This is a **medium** priority requirement.
Scope	This use case describes the process of a registered user accessing the Quotes section.
Precondition	The user is logged in to the SoulJournal application

Flow Description

Activation	This use case starts when a logged-in user wants to view inspirational quotes.
Main Flow	1. The system fetches quotes from the Quotable API. • The system displays the fetched quotes. • The user views the quotes. • The user can select the next quotes. • The user can select the previous quotes.

Alternate Flow

Title	**Description**
The user searches for quotes based on specific topics	A1.1. The system displays the filtered or searched quotes. A1.2. The use case continues at step 3 of the main flow.

Exceptional Flow

Title	Description
There is an error fetching quotes from the ZenQuotes API	E1.1. The system displays an error message. **(“There is an error displaying the quotes”)** E1.2. The user is prompted to try again. E1.3. The use case continues at step 1 of the main flow.

Post-Condition (For Successful Main Flow)

Title	Description
Termination	The user has viewed the quotes.

7. Provide Feedback

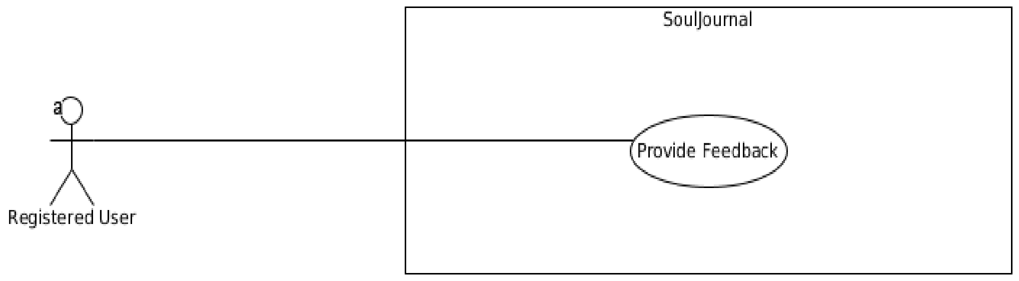


Figure 7.1: Provide Feedback

Name	**Provide Feedback**
Description	This use case describes the process of an user providing

	feedback, suggestions, or reporting issues about the SoulJournal application.
Priority	This is a **low** priority requirement.
Scope	This use case allows registered users to submit feedback, suggestions, or report issues regarding the SoulJournal application.
Precondition	The user is logged in to the SoulJournal application

Flow Description

Activation	This use case starts when a logged user wants to provide feedback, suggestions, or report an issue
Main Flow	1. The system displays the feedback form. • The user enters their feedback, suggestions, or issue details, followed by the amount of stars given. • The user submits the feedback or issue report. • The system displays a confirmation message

Alternate Flow

Title	**Description**
The user cancels the feedback submission	A1.1. The system discards any entered information A1.2. The use case terminates.

Exceptional Flow

Title	**Description**

There is an error processing the feedback or issue report.	E1.1. The system displays an error message. **(Title)** E1.2. The user is prompted to try again. E1.3. The use case continues at step 2 of the main flow.

Post-Condition (For Successful Main Flow)

Title	**Description**
Termination	The user's feedback, suggestions, or issue report has been submitted and processed by the system.

8. Settings

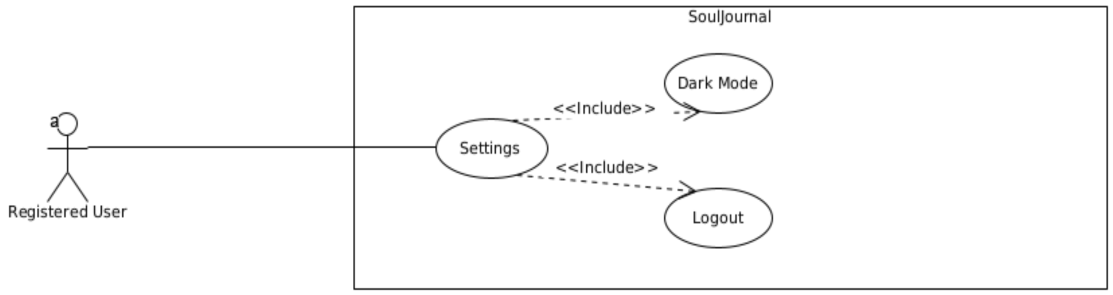


Figure 8.1: Settings

Name	**Settings (User Interface)**
Description	A registered user can access the settings of the SoulJournal application, such as themes and options
Priority	This is a **medium** priority requirement.

Scope	This use case describes the process of a registered user accessing the Settings option in the dashboard.
Precondition	The user is logged in to the SoulJournal application
Flow Description	
Activation	This use case starts when a logged-in user wants to customize the application's user interface or use another option.
Main Flow	1. The system displays the Settings options, including "Logout" and "Dark Mode". • The user selects the desired option. • The user applies the changes. • The system applies the user's choice.
Alternate Flow	
Title	**Description**
N/A	N/A
Exceptional Flow	
Title	**Description**
There is an error when applying the settings.	E1.1. The system displays an error message. **(Title)** E1.2. The user is prompted to try again or cancel the changes. E1.3. The use case continues at step 2 of the main flow.
Post-Condition (For Successful Main Flow)	

Title	Description
Termination	The system updates, taking into consideration the decision.

2.1.2 Data requirements

The SoulJournal application handles various types of data to provide a personalized journaling experience for users. Key data elements include user credentials(email and password) for registration and authentication, user preferences such as theme settings(Dark Mode), and journal entries. Additionally, the system manages feedback data to improve user experience and application functionality.

Data is transmitted between the frontend and backend using HTTP requests, typically in a RESTful API format. User actions on the frontend trigger these requests, sending data in a JSON format to the backend server. The server processes the requests, performs necessary operations such as database interactions, and sends back responses to the frontend.

2.1.3 User requirements

This section outlines the objectives and requirements for the SoulJournal web application from the customer’s perspective, detailing what users want and need from the system.

Objectives and Requirements

1. **Ease of Use:**
    - Users want an easy-to-navigate interface that will require minimal training and waiting time
    - Ability to customize their journaling experience.
2. **Accesibility:**

- Users want the ability to access their journals across various devices including desktops, laptops, and smartphones.

3. **Privacy and Security:**
   - Users require their journal entries to be securely stored and protected from unauthorized access.
4. **Functionality:**
   - Users want efficient ways to search categorize, and tag journal entries for easy retrieval
   - Users want a mini audio player that would be accessible in the creation of the entry.
5. **Inspiration and Motivation**
   - Users want something that will give them the motivation to write. Users will receive daily inspirational quotes from the Quotable API to motivate and inspire their journaling experience. This feature provides a fresh quote on a click of a button, and it also has the ability to retrieve previous quotes.

When the primary focus is on user needs, certain technical requirements must also be met to ensure the objectives are achieved:

- **Web Browser**: The SoulJournal web application will be accessible through a modern web browser, ensuring cross-platform compatibility.
- I**nternet Connection**: Users wil require an active internet connection to access the **SoulJournal** application, as it will be hosted on a remote server.
- **Computing Device:** users can access the SoulJournal application from various computed devices, such as desktop computers, laptops, or smart-phones

2.1.4 Environmental requirements

The SoulJournal application operates under the following environmental conditions:

1. **Server Environment:**
    - **Operating System:** The server that will host the backend is a Linux-based(LUbuntu) one.
    - **Database:** MongoDB is used for storing journal entries, user data, and feedback.
    - **Runtime Environment**: Node.js for executing server-side JavaScript code.
    - **Web Server:** Express.js for handling HTTP requests and serving the application.
2. **Client Environment:**
    - **Web Browsers:** The application is designed to be compatible with modern browser such as Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari.
    - **Operating Systems:** The application should function correctly on major operating systems, including Windows, macOS, and Linux.
3. **Development Environment:**
    - **IDE:** Visual Studio Code is used for development due to its robust support for JavaScript and Node.js
    - **Version Control**: Git for source code management and version control.
    - **Package Management:** npm for managing project dependencies.
4. **Hardware Requirements:**
    - **Server Hardware:** A running server with a bare minimum of resources (512MB RAM for the memory, 2GB for the storage should be enough).

- **Client Hardware:** Standard consumer-grade hardware(PCs, laptops, tablets) with sufficient processing power and memory to run modern web browsers.

2.1.5 Usability requirements

The SoulJournal web application must meet the following usability criteria to ensure a positive user experience:

1. **Ease of Use:**
    - The user interface should be intuitive and easy to navigate, allowing users to quickly access and use the application's features without extensive training or documentation.
    - Common acctions such as creating a new journal entry, logging in, and accessing settings should be easily accessible from the main dashboard.
2. **Consistency:**
    - The application sdhould maintain a consistent look and feel across all pages and components. This includes consistent use of colors, fonts, and button styles.
    - Error messages and notifications should follow a consistent format and be displayed in a predictable location.
3. **Performance:**
    - The application should load quickly and respond promptly to user interactions. Page load times should be minimized, and actions such as saving a journal entry or updating settings should be processed without noticeable delay.
    - The backend server should handle requests efficiently, ensuring a smooth user experience even under high load.
4. **Customization**

- Users should be able to customize their experience, such as toggling dark mode
- Customization options should be easily accessible from the settings page.

5. **Feedback**
    - The application sdhould provide immediate feedback for user actions. For example, when a user submits a journal entry, a confirmation message should be displayed.
    - Error messages should be clear and informative, guiding users on how to correct the issue.

2.2 Design and Architecture

The system architecture of SoulJournal is designed to be modular and scalable, ensuring smooth interaction between various components. The architecture will consist of three main layers:

- **Front-End**
- **Back-End**
- **Database**

Components:

1. **Front-End (Client-Side)**
    - **Technologies:** React.js, HTML, CSS, Bootstrap
    - **Role:** Provides the user interface for creating entries, managing entries, quote display, and feedback.
    - **Interaction:** Sends user inputs and requests to the backend via HTTP
2. **Back-End (Server-Side)**
    - **Technologies:** Node.js, Express.js

- **Role:** Handles authentication, user requests, and manages interactions with the database and external APIs.
- **Interaction:**
    - Receives data from the frontend.
    - Processes and validates the data.
    - Communicates with the database to store and retrieve data.
    - Integrates with the Quotable API to fetch and deliver quotes.
    - Sends responses back to the frontend with the necessary data.

3. **Database**

- **Technologies:** MongoDB
- **Role:** Stores user data, journal entries and feedback submissions.
- **Interaction:**
    - Receives read/write requests from the backend.
    - Returns the requested data to the backend for further processing or direct user interaction.

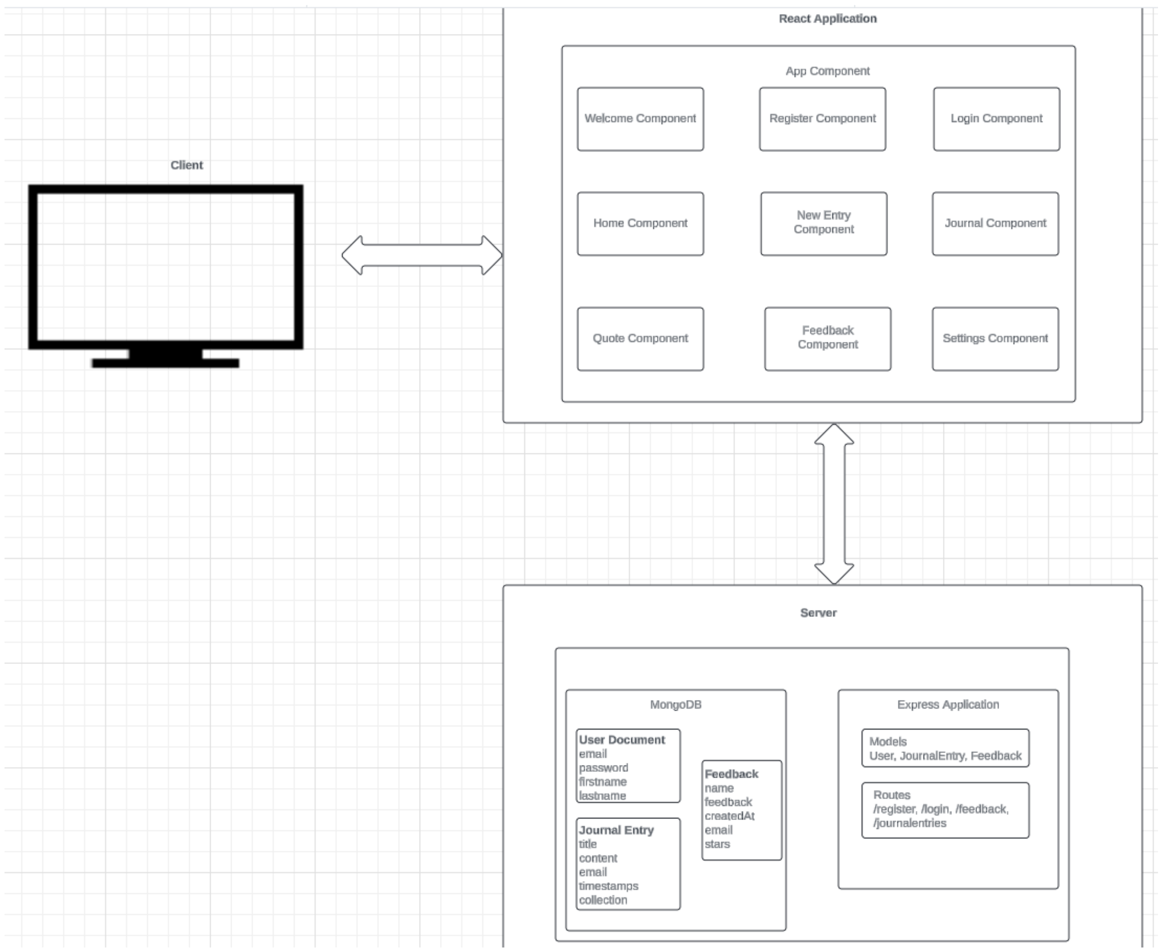


Figure 2.2.1: Architecture Design

1. **Client (React)**
   - **Components:**
     - **WelcomePage.js:** Serves as the landing page when the user enters the application. It includes the **Register** component and **Login** component as buttons.
     - **Register.js:** Handles user registration.
     - **Login.js:** Handles user authentication. Upon submission, it sends the credentials to the backend to verify and log the user in.

    - **Quotes.js:** This component displays quotes. It fetches quotes from an API.
    - **Feedback.js:** This component handles feedback submission. It includes a form where users can enter their comments or suggestions and submit them to the backend.
    - **NewEntry.js:** This component is used for creating new journal entries. It includes a form where users can write their thoughts or experiences, rate them according to their mood, and save them to the journal.
    - **Journal.js**: This component displays all the journal entries for the user. It fetches the entries from the backend and renders them for the user to view. The user can update the entries, and also delete them, by choice.
    - **Home.js**: This component serves as the main dashboard/homepage of the application. It provides navigation to other parts of the application.
    - **RadioPlayer.js:** This component is a radio player interface. It allows users to listen to live radio streams, depending on the type of music the user wants to listen.
    - **Settings.js:** This component allows users to manage their account settings. It includes options to change the password, logout or change the theme of the application.
  - **State Maangement:** Uses React’s useState and useEffect hooks.
  - **Styling:** Uses Bootstrap for styling.
2. **Server (Node.js)**
  - **Endpoints:**
    - /auth/register*:* Handles user registration.
    - /auth/login: Handles user authentication

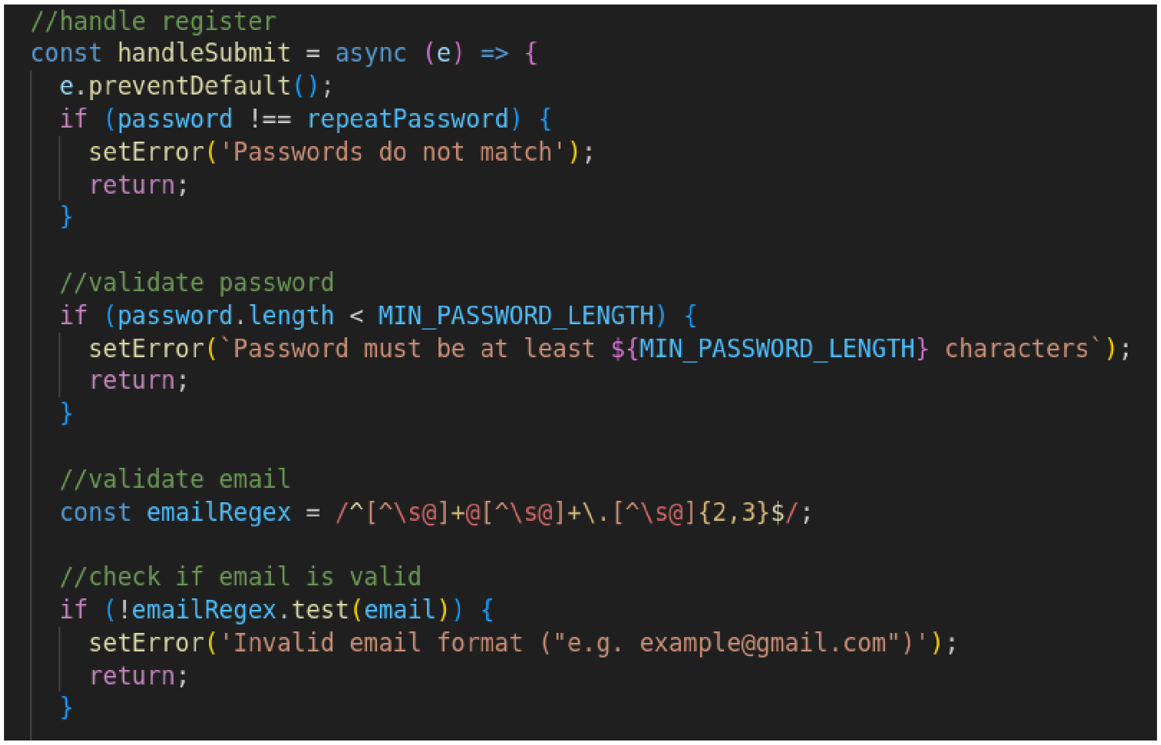
        - journal/new_entry: Handles new entry creation.
        - journal/entries: Handles R.U.D requests(Read, Update, Delete).
        - /feedback/submit-feedback: Handles feedback submissions.
    - Middleware: Uses Express.js for routing and middleware, Cors for middleware.

3. **Database (MongoDB)**
    - **Collections:**
        - users: Stores user information.
        - entries: Stores journal entries.
        - feedback: Stores user feedback.

Main Algorithms

1. **User Registration**

   The user registration process involves validating the email format and ensuring the password meets a minimum length requirement.



```
//handle register
const handleSubmit = async (e) => {
  e.preventDefault();
  if (password !== repeatPassword) {
    setError('Passwords do not match');
    return;
  }

  //validate password
  if (password.length < MIN_PASSWORD_LENGTH) {
    setError(`Password must be at least ${MIN_PASSWORD_LENGTH} characters`);
    return;
  }

  //validate email
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]{2,3}$/;

  //check if email is valid
  if (!emailRegex.test(email)) {
    setError('Invalid email format ("e.g. example@gmail.com")');
    return;
  }
```

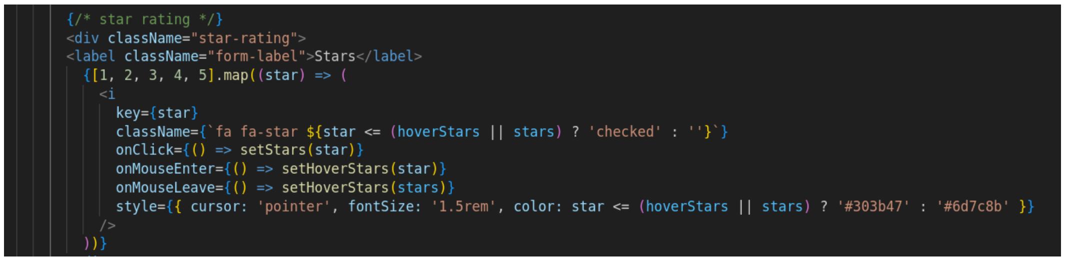
Figure 1.1: User Registration Algorithm

Algorithm:

1. Validate email format using a regular expression.
2. Check if the password meets the minimum length requirement.
3. Check if the **password** matches with the **repeatPassword**
4. If valid, send a POST request to the server to register the user.

2. Star Rating System

The star rating system allows users to rate from 1 to 5 stars. The rating is updated based on user interaction (click and hover).



```
{/* star rating */}
<div className="star-rating">
<label className="form-label">Stars</label>
  {[1, 2, 3, 4, 5].map((star) => (
    <i
      key={star}
      className={`fa fa-star ${star <= (hoverStars || stars) ? 'checked' : ''}`}
      onClick={() => setStars(star)}
      onMouseEnter={() => setHoverStars(star)}
      onMouseLeave={() => setHoverStars(stars)}
      style={{ cursor: 'pointer', fontSize: '1.5rem', color: star <= (hoverStars || stars) ? '#303b47' : '#6d7c8b' }}
    />
  ))}
```

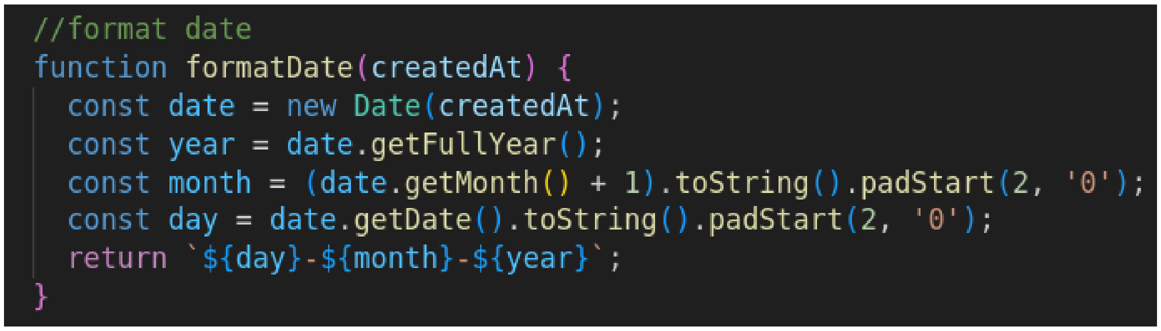
Figure 2.1: Star Rating System

Algorithm:

1. Initialize state variables for **stars** and **hoverStars**.
2. On star click, update the **stars** state.
3. On star hover, update the **hoverStars** state.
4. On mouse leave, reset **hoverStars** to **stars**.

3. Date Formatting Algorithm

An algorithm that converts a date string into a **Date** object, extracts year, month, day, and formats it as DD-MM-YYYY.



```
//format date
function formatDate(createdAt) {
  const date = new Date(createdAt);
  const year = date.getFullYear();
  const month = (date.getMonth() + 1).toString().padStart(2, '0');
  const day = date.getDate().toString().padStart(2, '0');
  return `${day}-${month}-${year}`;
}
```

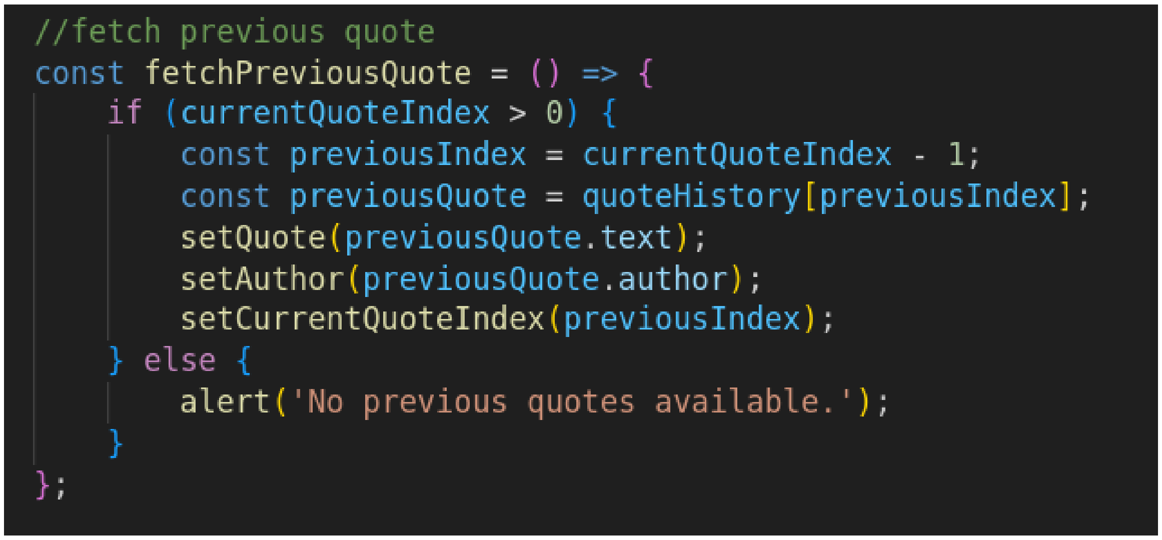
Figure 3.1: formatDate Algorithm

Algorithm:

1. Define a function formatDate that takes a parameter createdAt.
2. Inside the function, create a new Date object using the createdAt parameter.
3. Extract the year, month, and day using getFullYear(), getMonth() + 1, getDate() and pad the last two get methods to only two digits.
4. Combine the day, month, and year into a string formatted as DD-MM-YYYY.
5. Return the formatted date string from the function.

4. **Fetch Previous Quote Algorithm**

This function allows the user to navigate to the previous quote in the **quoteHistory** array, updating the displayed quote and author. If there are no previous quotes, it alerts the user.



```
//fetch previous quote
const fetchPreviousQuote = () => {
    if (currentQuoteIndex > 0) {
        const previousIndex = currentQuoteIndex - 1;
        const previousQuote = quoteHistory[previousIndex];
        setQuote(previousQuote.text);
        setAuthor(previousQuote.author);
        setCurrentQuoteIndex(previousIndex);
    } else {
        alert('No previous quotes available.');
    }
};
```

Figure 4.1: fetchPreviousQuote Algorithm

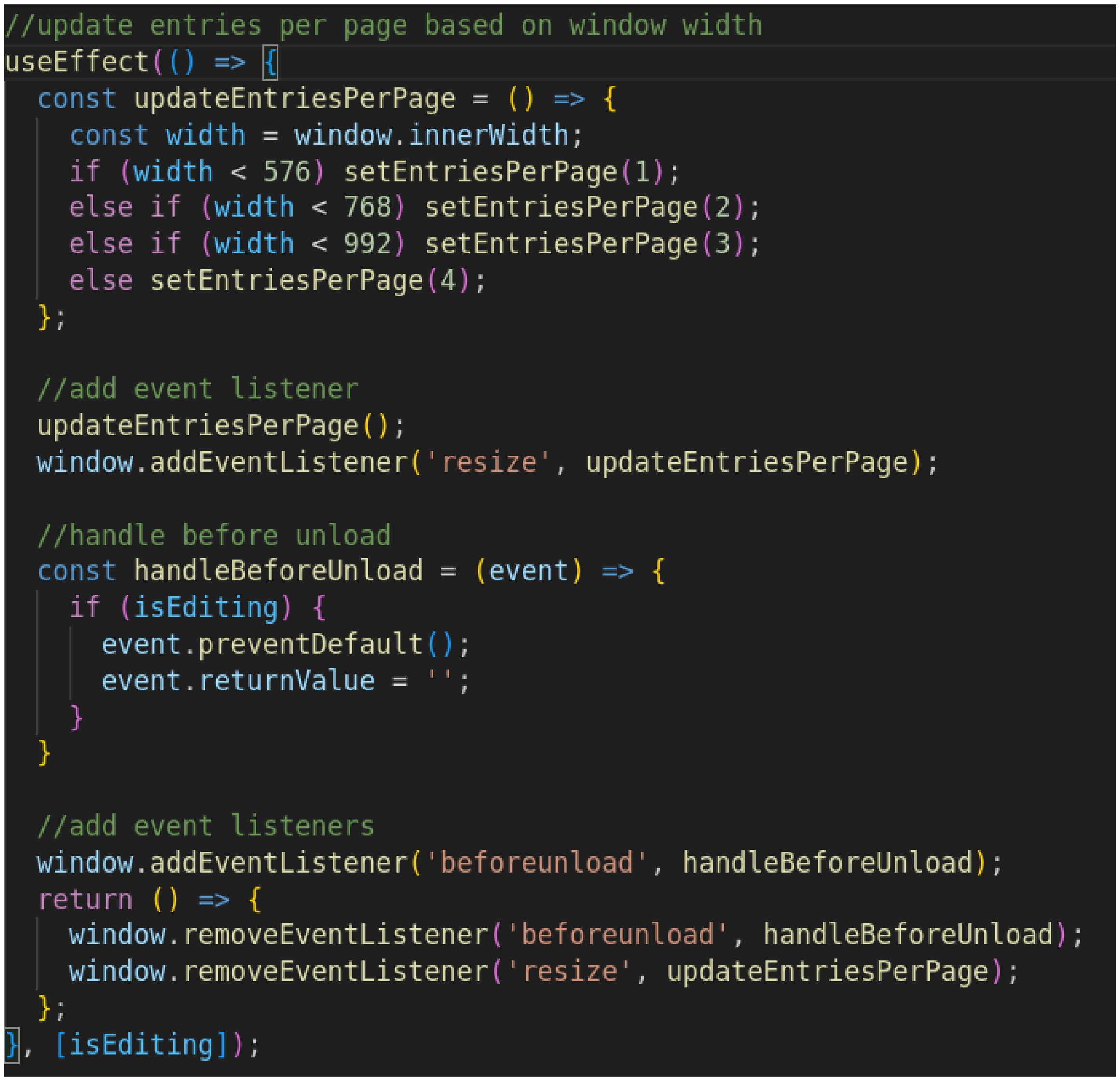
Algorithm:

1. The function first checks if the **currentQuoteIndex** is greather than 0. This ensures that there is at least one previous quote available in the **quoteHistory.**

2. If there is a previous quote, it calculates the **previousIndex** by substracting 1 from the **currentQuoteIndex**.
3. The function updates the state with the text and author of the **previousQuote** using **setQuote** and **setAuthor** respectively. It will also update the **currentQuoteIndex** to the **previousIndex** using **setCurrentQuoteIndex**.
4. If the **currentQuoteIndex** is not greater than 0, it means there are no previous quotes available. In this case, the function displays an alert with the message “No previous quotes available.”

5. Entries Per Page Logic Algorithm

An algorithm that dynamically adjusts the number of journal entries displayed per page based on the window width.



```
//update entries per page based on window width
useEffect(() => {
  const updateEntriesPerPage = () => {
    const width = window.innerWidth;
    if (width < 576) setEntriesPerPage(1);
    else if (width < 768) setEntriesPerPage(2);
    else if (width < 992) setEntriesPerPage(3);
    else setEntriesPerPage(4);
  };

  //add event listener
  updateEntriesPerPage();
  window.addEventListener('resize', updateEntriesPerPage);

  //handle before unload
  const handleBeforeUnload = (event) => {
    if (isEditing) {
      event.preventDefault();
      event.returnValue = '';
    }
  }

  //add event listeners
  window.addEventListener('beforeunload', handleBeforeUnload);
  return () => {
    window.removeEventListener('beforeunload', handleBeforeUnload);
    window.removeEventListener('resize', updateEntriesPerPage);
  };
}, [isEditing]);
```

Figure 4.1: Entries per Page

Algorithm:

1. The **updateEntriesPerPage** function determines the number of entries to display per page based on the current window width. It also uses **window.innerWidth** to get the current width of the window.
2. Set the number of entries per page based on window width.

3. The **updateEntriesPerPage** function is called immediately to set the initial number of entries per page based on the current window width.
4. An event listener is added to the **resize** event of the window. Whenever the window is resized, the **updateEntriesPerPage** function is called to adjust the number of entries per page dynamically.

2.3 Implementation

Below I have selected the main pages that I've considered to be interesting in terms of interesting code snippets.

Home.js

The **Home** page is a main component of the web application. It serves as a dashboard for the user, each button being a link for a different component.

Main functions

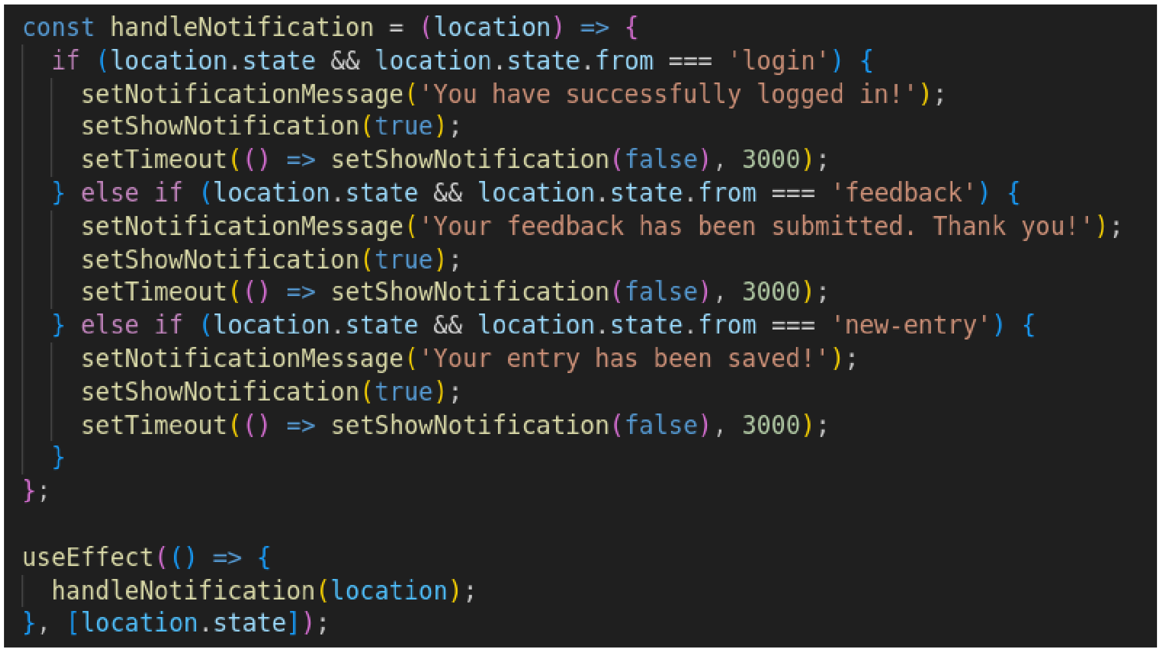
getUserDetails(): It is an asynchronous function that is designed to:

- Retrieve the user's email from local storage.
- Fetch user details from the backend using the retrieved email.
- Update the state with the fetched user details.
- Save the fetched user details back to the local storage.

Interesting code snippet:

- **handleNotification function**

This function is responsible for displaying different notification messages based on the state of the location object. The **useEffect** hook ensures that the **handleNotification** function is called whenever the **location.state** changes, thereby updating the notification accordingly.



```
const handleNotification = (location) => {
  if (location.state && location.state.from === 'login') {
    setNotificationMessage('You have successfully logged in!');
    setShowNotification(true);
    setTimeout(() => setShowNotification(false), 3000);
  } else if (location.state && location.state.from === 'feedback') {
    setNotificationMessage('Your feedback has been submitted. Thank you!');
    setShowNotification(true);
    setTimeout(() => setShowNotification(false), 3000);
  } else if (location.state && location.state.from === 'new-entry') {
    setNotificationMessage('Your entry has been saved!');
    setShowNotification(true);
    setTimeout(() => setShowNotification(false), 3000);
  }
};

useEffect(() => {
  handleNotification(location);
}, [location.state]);
```

Figure 2.3.1: handleNotification

1. **Parameters**

   - **location**: This is an object provided by the **useLocation** hook. It contains information about the current URL, including any state passed via navigation

2. **Logic**

   - The function checks if **location.state** exists and then examines the **from** property within **location.state**.
   - Depending on the value of **location.state.from**, it sets a corresponding notification message and shows the notification for three seconds.

3. **Notification Handling**

   - **setNotificationMessage**: Sets the message to be displayed in the notification.
   - **setShowNotification**: Controls the visibility of the notification.
   - **setTimeout**: Hides the notification after three seconds

Journal.js

The **Journal** page is where all the **R.U.D**(Read, Update, Delete) operations happen. On this page we can see the journal entries, edit the entries, or delete them, by choice.

Main functions

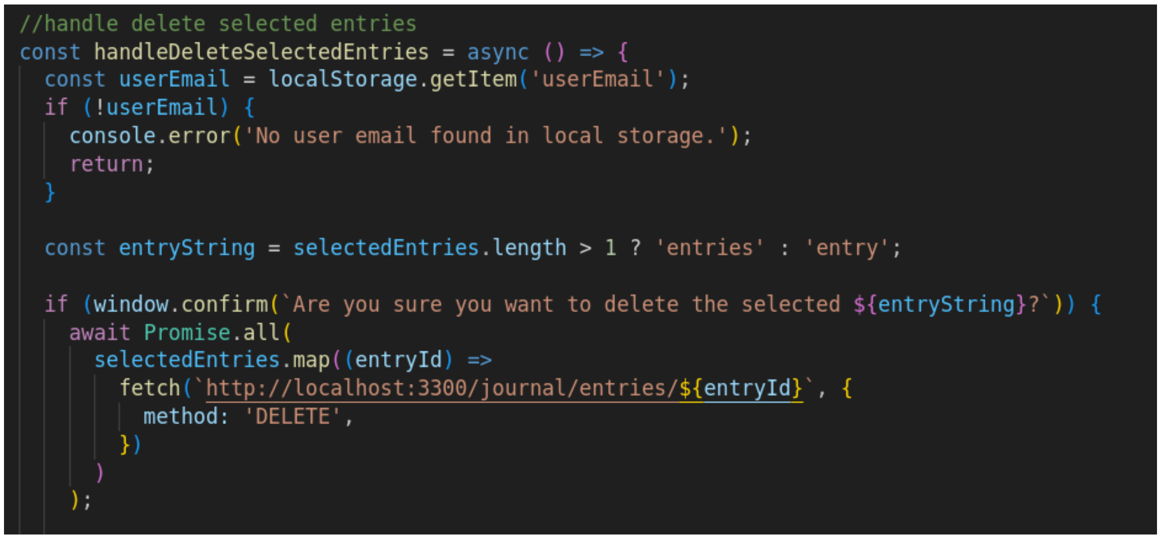
handleSave():

- Retrieves user email from local storage.
- Confirms update with the user.
- Sends a **PUT** request to update the current journal entry.
- If successful:
    - Updates the entry in the state.
    - Displays a notification.
    - Closes editing and modal.
- Logs an error if the update fails.

Interesting code snippet

Although it is just a line of code that is interesting, in my opinion, makes the whole thing look nice.

We have the **handleDeleteSelectedEntries** function that, if the Delete mode is activated, will permanently delete the selected entries. If there are entries selected, the web application will display an alert, with the message `Are you sure you want to delete the selected ${**entryString**}?`, where **entryString** will be consisting of the word "**entry**" or "**entries**", depending on the number of entries selected.



```javascript
//handle delete selected entries
const handleDeleteSelectedEntries = async () => {
  const userEmail = localStorage.getItem('userEmail');
  if (!userEmail) {
    console.error('No user email found in local storage.');
    return;
  }

  const entryString = selectedEntries.length > 1 ? 'entries' : 'entry';

  if (window.confirm(`Are you sure you want to delete the selected ${entryString}?`)) {
    await Promise.all(
      selectedEntries.map((entryId) =>
        fetch(`http://localhost:3300/journal/entries/${entryId}`, {
          method: 'DELETE',
        })
      )
    );
```

Figure 2.3.2: entryString

I know it is a relatively simple line of code, but small things brought together make the web application better.

Register.js

The Register page is used when a visitor wants to create an account on the web application. We have added some control measures for the passwords and the email, so that the user must follow the rules regarding the email address entered (must be a valid email address). For the password, I have added a minimum of five characters as it was easier for the testing, even though the right syntax would be at least eight characters, an uppercase, a lowercase, and a special character. For the email, password, and repeat password, I have also added an alert so that the web application can alert the user when the details entered are correct / incorrect.

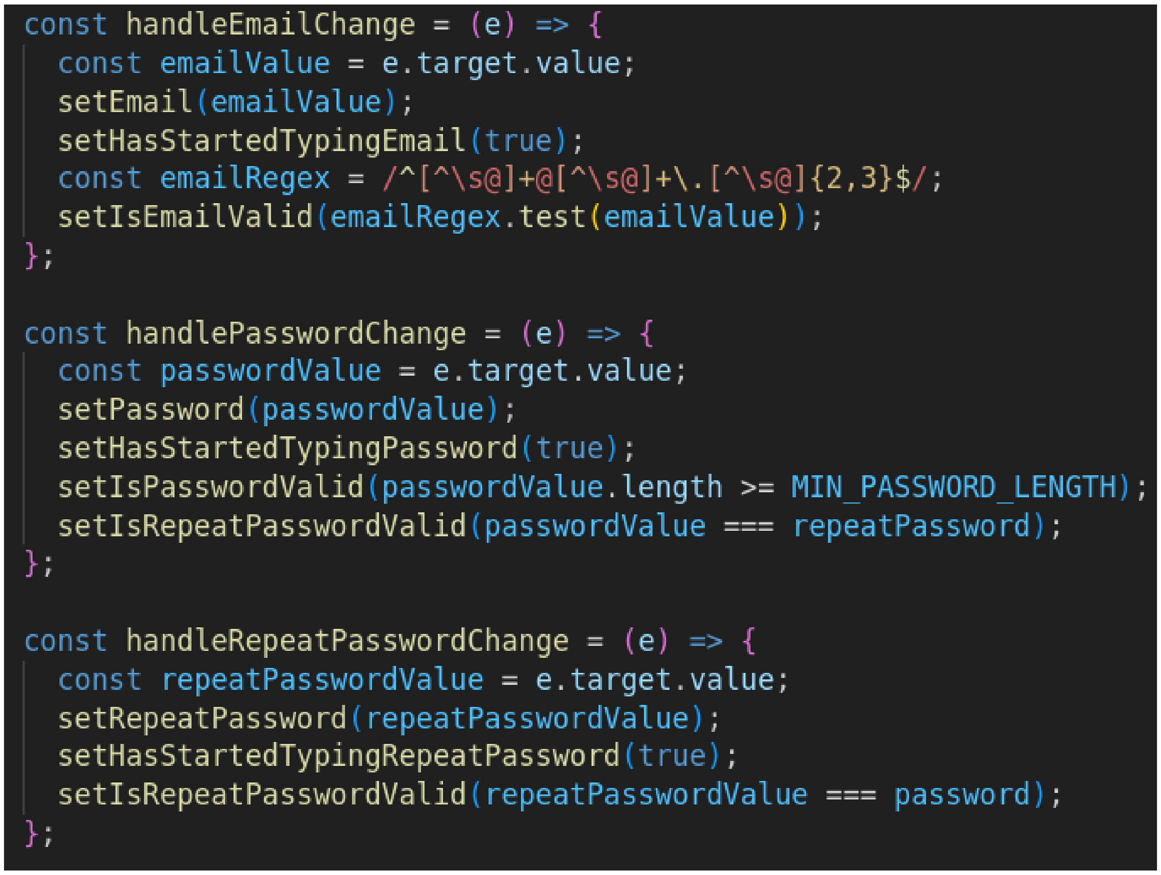
Main function

handleSubmit():

- Prevents default form submission behaviour.
- Checks if passwords match, if not it will set an error.
- Validates password length, it will set an error if it is too short.

- Validates email format using a regex, it will set an error if it is too short.
- Sends a POST request to register the user.
- If successful, navigates to the registration success page.
- If registration fails, it will set an error message.
- Catches and handles any errors during the process.

An interesting code snippet would be this one:



```
const handleEmailChange = (e) => {
  const emailValue = e.target.value;
  setEmail(emailValue);
  setHasStartedTypingEmail(true);
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]{2,3}$/;
  setIsEmailValid(emailRegex.test(emailValue));
};

const handlePasswordChange = (e) => {
  const passwordValue = e.target.value;
  setPassword(passwordValue);
  setHasStartedTypingPassword(true);
  setIsPasswordValid(passwordValue.length >= MIN_PASSWORD_LENGTH);
  setIsRepeatPasswordValid(passwordValue === repeatPassword);
};

const handleRepeatPasswordChange = (e) => {
  const repeatPasswordValue = e.target.value;
  setRepeatPassword(repeatPasswordValue);
  setHasStartedTypingRepeatPassword(true);
  setIsRepeatPasswordValid(repeatPasswordValue === password);
};
```

Figure 2.3.3: handleValueChange

handleEmailChange():

- Retrieves the email value from the event target.
- Updates the email state.
- Sets the flag indicating the user has started typing the email.

- Defines a regex for email validation.
- Updates the email validity state.

handlePasswordChange():

- Retrieves the password value from the event target.
- Updates the password state.
- Sets the flag indicating the user has started typing the password.
- Updates the password validty state based on the minimum length requirement.
- Updates the repeat password validty state by comparing it with the current password.

handleRepeatPasswordChange:

- Retrieves the repeat password value from the event target.
- Updates the repeat password state.
- Sets the flag indicating the user has started typing the repeat password.
- Updates the repeat password validty state by comparing it with the current password.

2.4 Testing

With less than 10 minutes until the submission, I don’t have enough time to post any of the tests that I’ve done. The only one that I managed to implement were the manual ones, like doing a journal entry 20 times, to see the aspect of the page, or checking the login page and the register page for flaws in the forms.

2.5 Graphical User Interface (GUI) Layout

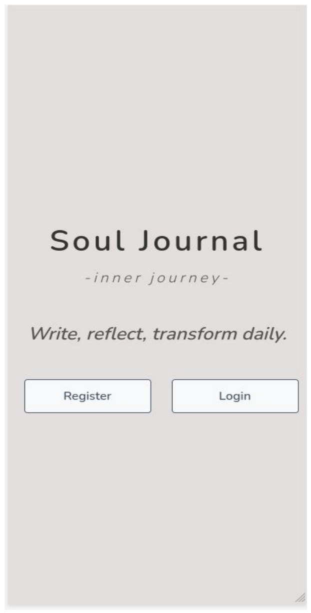


Figure 2.5.1: Welcome Page

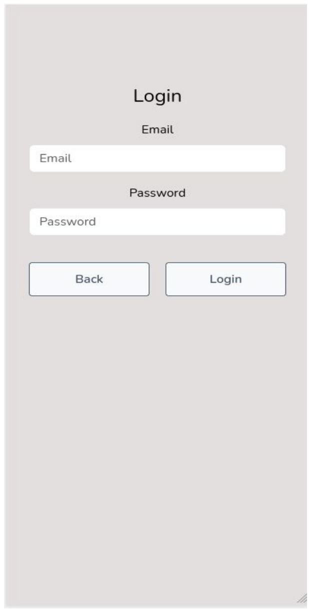


Figure 2.5.2: Login Page

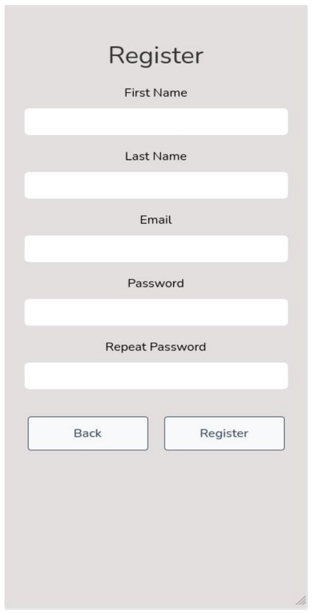


Figure 2.5.3: Register Page

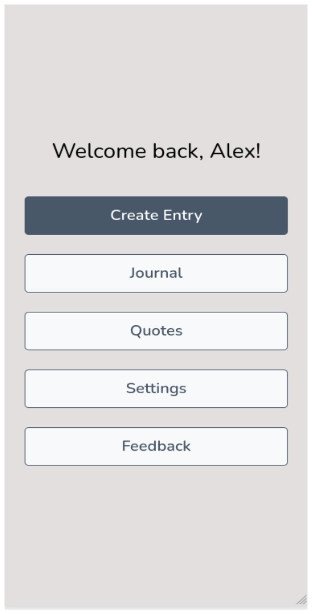


Figure 2.5.4: Home Page

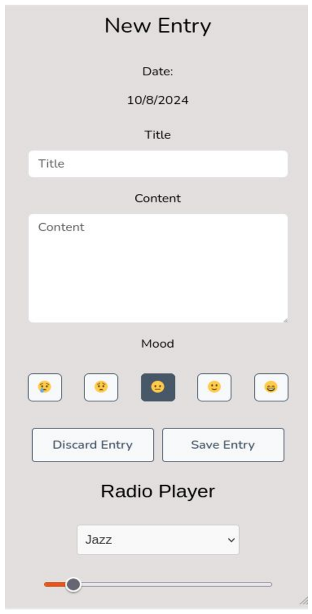


Figure 2.5.5: New Entry Page

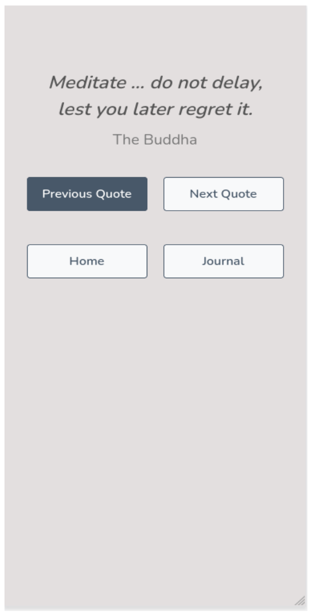


Figure 2.5.6: Quotes Page

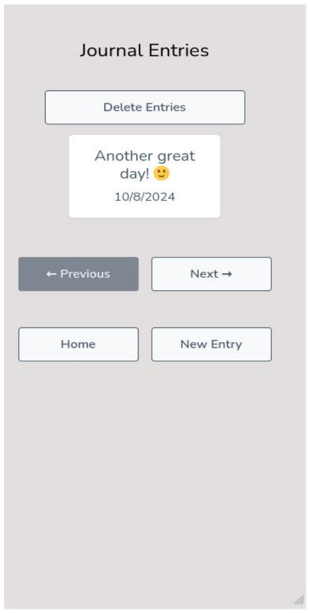
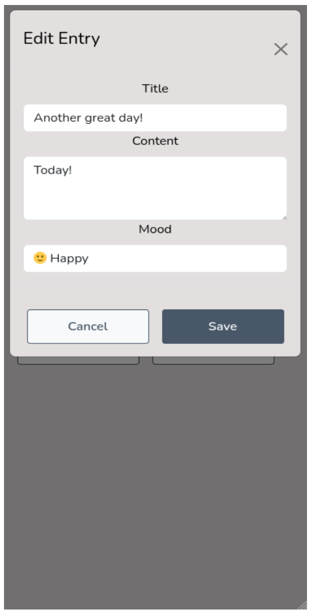


Figure 2.5.7: Journal Page



Title: Another great day!

Content: Today!

Mood: 🙂 Happy

Figure 2.5.8: Edit Entry Page

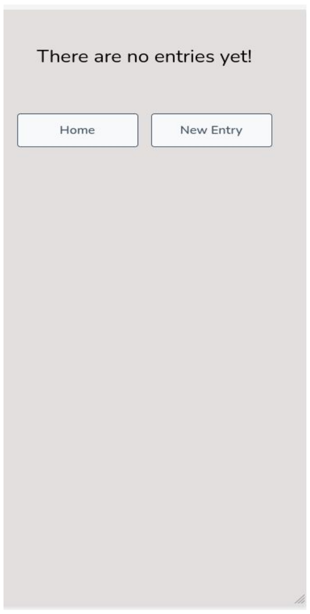


Figure 2.5.9: Empty Journal Page

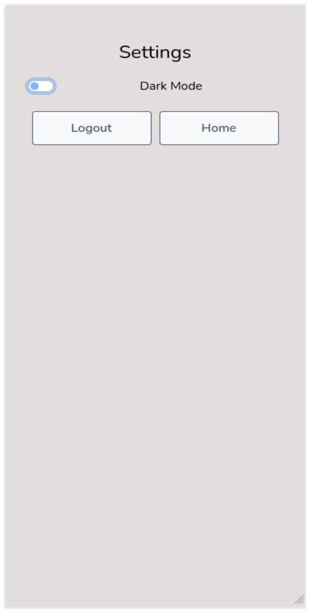


Figure 2.5.10: Settings Page

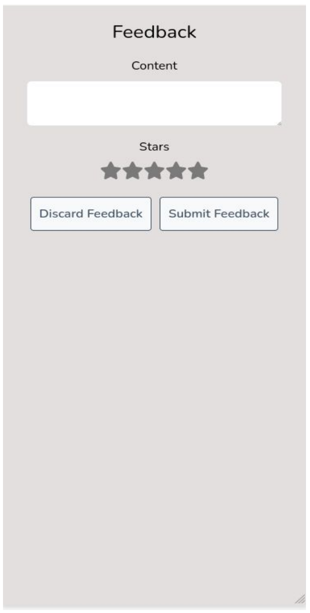


Figure 2.5.11: Feedback Page

The **SoulJournal** web application was designed to have a minimalist layout. I consider the application to be a user-friendly and intuitive one, focusing on the sole purpose of the user to write his thoughts.

1. **Welcome Page (Figure 2.5.1)**

   It serves as the landing page for visitors when they visit the web application. It displays a welcome message and the **SoulJournal** logo.

2. **Login Page (Figure 2.5.2.)**

   The login page allows users to enter their email and password to access their accounts. It features a form with input fields for email and password, and a submit button. Upon submission, it sends the credentials to the server for authentication. If successful, the user is navigated to the appropriate page, otherwise an error message is displayed.

3. **Register Page (Figure 2.5.3)**

   The registration page allows visitors to create a new account by entering their email and password. It includes input fields for email, password, and repeat password. The page validates the email format and checks if the password meets a minimum length requirement. As users type, it provides real-time feedback on the validity of their inputs. Upon successful validation, users can submit the form to register their account.

4. **Home Page (Figure 2.5.4)**

   The home page serves as the main dashboard for logged-in users. It displays a welcome message and provides access to various features and sections of the application.

5. **New Entry Page (Figure 2.5.5)**

   The new entry page allows users to create a new journal entry. It features a form with fields for the entry date (auto-filled with the current date) and a title. Users can input a title for their entry, which is required. Upon submission, the form data is processed to create the new entry.

6. **Quotes Page (Figure 2.5.6)**

   The quotes page displays a random quote to the user. It fetches a new quote from an external API when the page loads. The page includes a quote and its author, and it maintains a history of previously fetched quotes. Users can navigate through the quote history to view past quotes.

7. **Journal Page (Figure 2.5.7)**

   The journal page allows users to view and manage their journal entries. It displays a list of entries, each within a card that shows the entry's details. It has the option to update entries and also to delete them.

8. **Edit Entry Page (Figure 2.5.8)**

   The edit entry page allows users to modify an existing journal entry. It features a form pre-filled with the entry's current details, including the title and content. Users can update these fields and submit the form to save changes. The page ensures a seamless editing experience.

9. **Empty Journal Page (Figure 2.5.9)**

   The empty journal page displays a message indicating no entries are available. It encourages users to create their first entry.

10. **Settings Page (Figure 2.5.10)**

    The settings page allows users to toggle dark mode on and off. It features a switch to enable or disable dark mode, updating the page's appearance and saving the preference in local storage.

11. **Feedback Page (Figure 2.5.11)**

    The feedback page allows users to submit their feedback and rate their experience with stars. It includes a form for entering feedback and selecting a star rating. Users can submit their feedback or discard it, which clears the form and navigates back to the home page.

2.6 Customer testing

The application was test by family members, respectively my partner and my cousins. Below I have attached screenshots with the member testing the application and the feedback given.

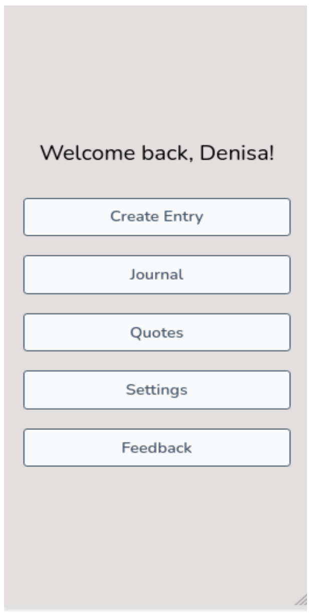


Figure 2.6.1: Denisa/Dashboard

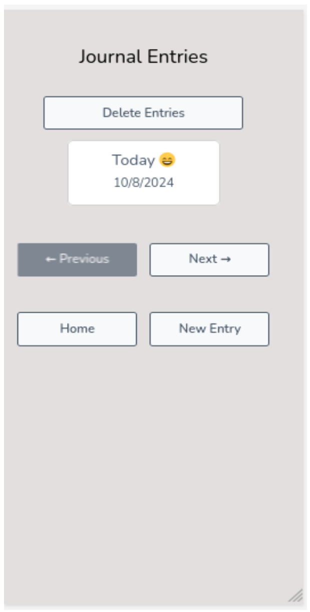
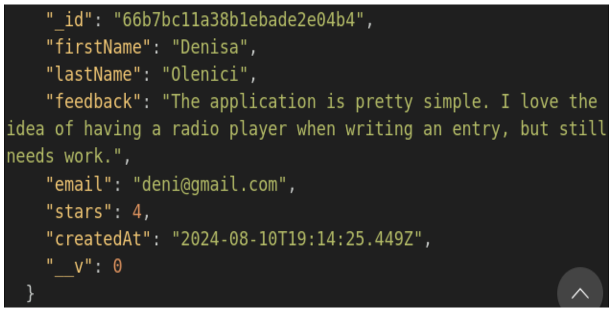


Figure 2.6.2 Denisa/Journal



```
    "_id": "66b7bc11a38b1ebade2e04b4",
    "firstName": "Denisa",
    "lastName": "Olenici",
    "feedback": "The application is pretty simple. I love the
idea of having a radio player when writing an entry, but still
needs work.",
    "email": "deni@gmail.com",
    "stars": 4,
    "createdAt": "2024-08-10T19:14:25.449Z",
    "__v": 0
  }
```

Figure 2.6.3: Denisa/Feedback

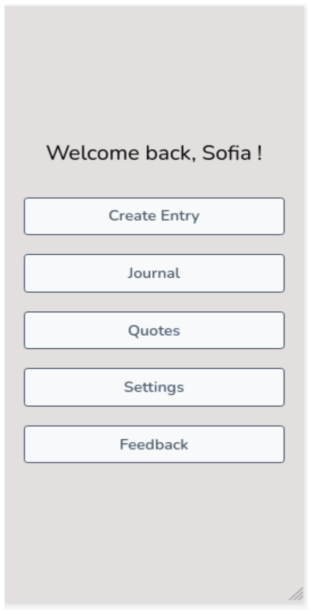
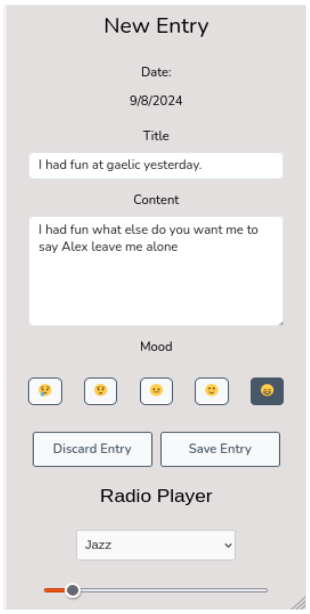


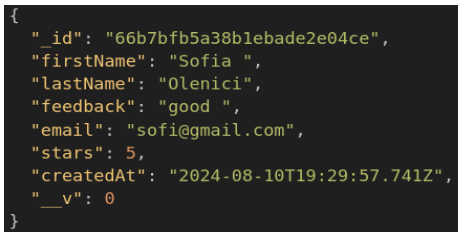
Figure 2.6.4: Sofia/Dashboard



Title: I had fun at gaelic yesterday.

Content: I had fun what else do you want me to say Alex leave me alone

Figure 2.6.5: Sofia/New Entry



```
{
  "_id": "66b7bfb5a38b1ebade2e04ce",
  "firstName": "Sofia ",
  "lastName": "Olenici",
  "feedback": "good ",
  "email": "sofi@gmail.com",
  "stars": 5,
  "createdAt": "2024-08-10T19:29:57.741Z",
  "__v": 0
}
```

Figure 2.6.6: Sofia/Feedback

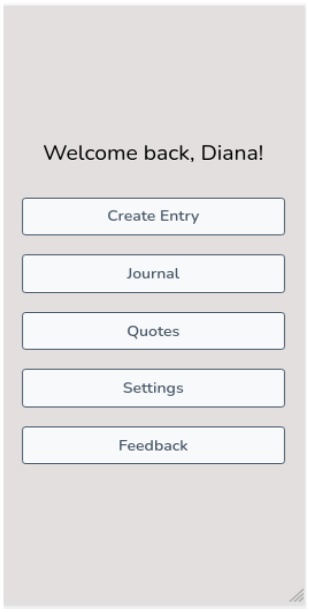


Figure 2.6.7: Diana/Dashboard

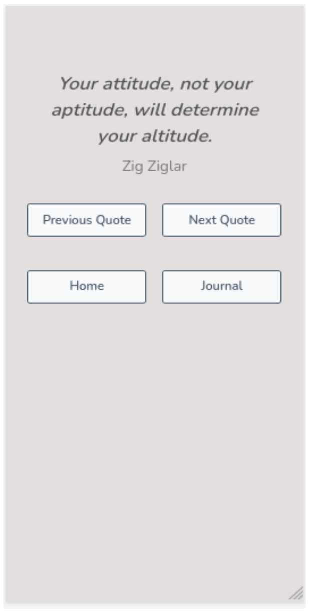
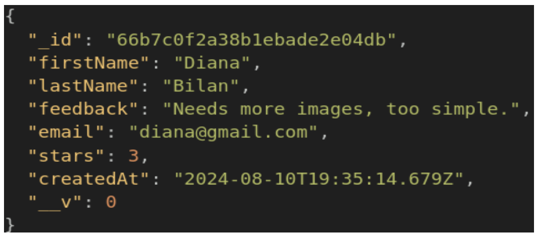

Your attitude, not your aptitude, will determine your altitude.
Zig Ziglar
Previous Quote
Next Quote
Home
Journal

Figure 2.6.8: Diana/Quotes



```
{
  "_id": "66b7c0f2a38b1ebade2e04db",
  "firstName": "Diana",
  "lastName": "Bilan",
  "feedback": "Needs more images, too simple.",
  "email": "diana@gmail.com",
  "stars": 3,
  "createdAt": "2024-08-10T19:35:14.679Z",
  "__v": 0
}
```

Figure 2.6.9: Diana/Feedback

2.7 Evaluation

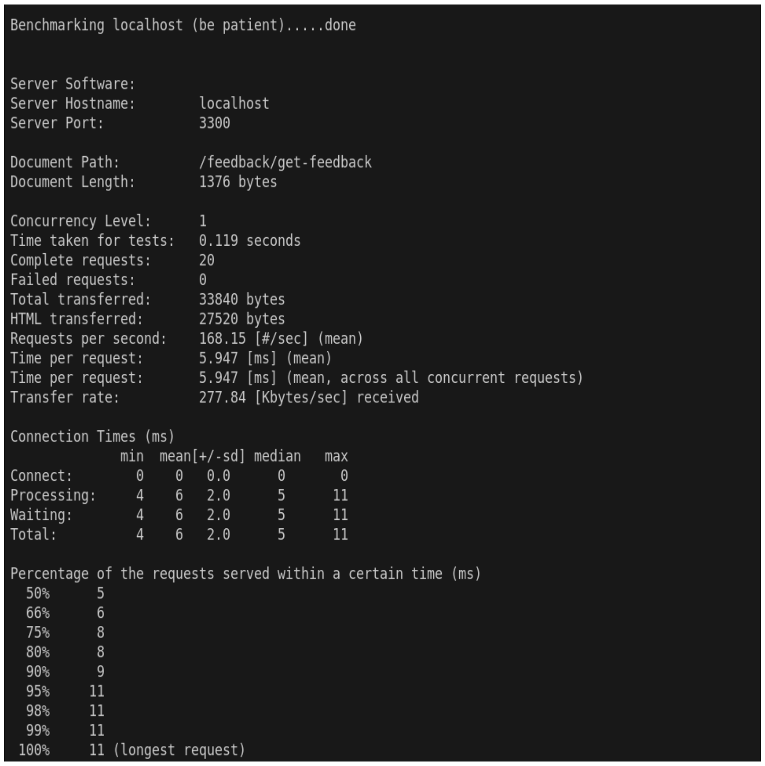
The system was evaluated through a combination of user feedback and performance metrics. User feedback was collected via a feedback submission feature integrated into the application. This feature allowed users to rate their experience on a scale of one to five stars and provide written comments. The feedback received included three ratings of 3, 4, and 5 stars, respectively.

One user rated the application four stars and commented, “The application is pretty simple. I love the idea of having a radio player when writing an entry, but still needs work.” This feedback highlights the appreciation for the radio player feature but also indicates that there is room for improvement in the overall application complexity and functionality.

Another user provided a five star rating with the comment “good”, suggesting a positive experience.

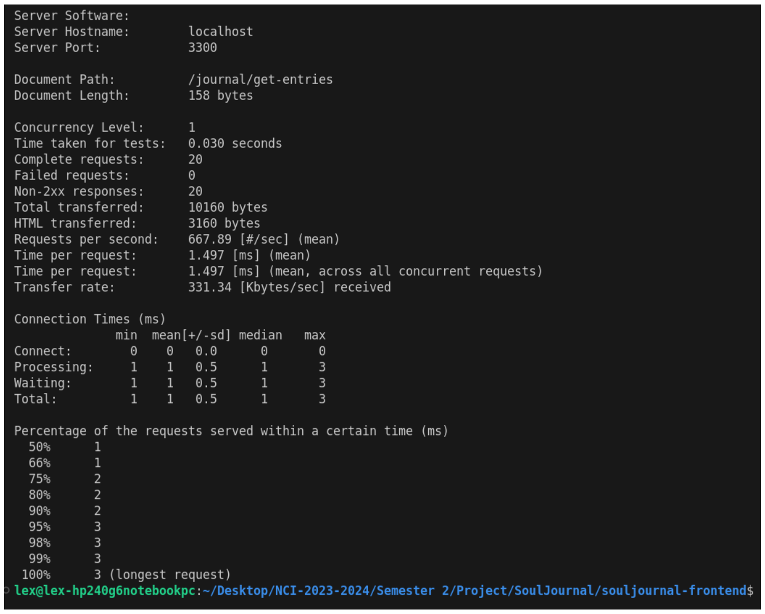
The third user gave a three star rating and mentionet, “Needs more images, too simple.” This feedback suggests that the application could benefit from additional visual elements to enhance the user experience.

In terms of performance, the system was evaluated for scalability and correctness. The backend API endpoints were tested for response times. The tests were done using Apache Benchmark, for the Feedback (Figure 2.7.1), and Journal Entries(Figure 2.7.2)



```
Benchmarking localhost (be patient).....done


Server Software:
Server Hostname:        localhost
Server Port:            3300

Document Path:          /feedback/get-feedback
Document Length:        1376 bytes

Concurrency Level:      1
Time taken for tests:   0.119 seconds
Complete requests:      20
Failed requests:        0
Total transferred:      33840 bytes
HTML transferred:       27520 bytes
Requests per second:    168.15 [#/sec] (mean)
Time per request:       5.947 [ms] (mean)
Time per request:       5.947 [ms] (mean, across all concurrent requests)
Transfer rate:          277.84 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:     4    6   2.0      5      11
Waiting:        4    6   2.0      5      11
Total:          4    6   2.0      5      11

Percentage of the requests served within a certain time (ms)
  50%      5
  66%      6
  75%      8
  80%      8
  90%      9
  95%     11
  98%     11
  99%     11
 100%     11 (longest request)
```

Figure 2.7.1: Feedback GET Request



```
Server Software:
Server Hostname:        localhost
Server Port:            3300

Document Path:          /journal/get-entries
Document Length:        158 bytes

Concurrency Level:      1
Time taken for tests:   0.030 seconds
Complete requests:      20
Failed requests:        0
Non-2xx responses:      20
Total transferred:      10160 bytes
HTML transferred:       3160 bytes
Requests per second:    667.89 [#/sec] (mean)
Time per request:       1.497 [ms] (mean)
Time per request:       1.497 [ms] (mean, across all concurrent requests)
Transfer rate:          331.34 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.0      0       0
Processing:     1    1   0.5      1       3
Waiting:        1    1   0.5      1       3
Total:          1    1   0.5      1       3

Percentage of the requests served within a certain time (ms)
  50%      1
  66%      1
  75%      2
  80%      2
  90%      2
  95%      3
  98%      3
  99%      3
 100%      3 (longest request)
lex@lex-hp240g6notebookpc:~/Desktop/NCI-2023-2024/Semester 2/Project/SoulJournal/souljournal-frontend$
```

Figure 2.7.2: Entries Request

Overall, the evaluation results indicate a positive user experience with specific areas identified for further enhancement, particularly in terms of visual appeal and additional features.

3 Conclusions

Advantages:

- **User-Friendly Interface: SoulJournal** offers a simple and intuitive interface, making it easy for uers to document their thoughts and experiences.
- **Unique Features:** The integration of a radio player enhances the journaling experience, providing a calming background while writing.
- **Positive User Feedback:** Users have expressed satisfaction with the application's core functionalities, indicating a soid foundation for further development.

Disadvantages:

- **Simplicity:** Some users find the application too simple, suggesting a need for additional features and visual enhancements.
- **Limited Visual Appeal:** Feedback indicates a desire for more images and visual elements to make the application more engaging.

Opportunities:

- **Feature Expansion:** There is potential to add more features, such as mood tracking, image uploads, and customziable themes, to enhance user engagement.
- **User Feedback Integration:** Leveraging user feedback to continuously improve the application can lead to higher user satisfaction and retention.
- **Scalability:** The application can be scaled to support a larger user base, with potential for cloud integration to handle increased data storage and processing needs.

Limits:

- **Resource Constraints:** Development and enhancement of new features may be limited by available resources, such as time and budget.
- **Technical Challenges:** Ensuring the application remains responsive and reliable as new features are added may present technical challenges.
- **User Adoption:** Attracting and retaining a larger user base requires effective marketing and continuous improvement based on user feedback.
- **Lack of Knowledge in Certain Frameworks:** With limited experience with specific frameworks or libraries, that could hinder the implementation of advanced features and optimizations.

4 Further development or research

With additional resources, the **SoulJournal** project has the potential to offer enhanced features and capabilities:

Advanced Features

- **Mood Tracking:** Implementing mood tracking with visual analytics to help users understand their emotional patterns over time.
- **Image and Media Integration:** Allowing users to upload images, audio, and videos to enrich their journal entries.
- **Customizable Themes:** Providing various themes and customization options to personalize the journaling experience.

Enhanced User Experience

- **Mobile Application:** Developing an offline version of **SoulJournal** for both iOS and Android to provide users with access even though there's no internet, being able to create entries, and for the app to wait for internet access to have them posted.
- **Voice-to-Text:** Integrating voice-to-text functionality to allow users to dictate their entries, making journaling more accessible.
- **AI-Powered Insights:** Utilizing AI to offer insights and suggestions based on the user's entries, such as identifying recurring themes or providing motivational quotes.

Scalability and Performance

- **Cloud Integration:** Migrating to cloud-based infrastructure to handle increased data storage and processing needs, ensuring scalability and reliability.
- **Real-Time Collaboration:** Introducing Features for real-time collaboration, allowing users to share and co-author entries with trusted individuals.

5 References

Gregory Ciotti (2023). The Psychological Benefits of Writing. (25/06/2024)

Adam Grant, Jane Dutton (2012). Beneficiary or Benefactor: Are People More Prosocial When They Reflect on Receiving or Giving? (25/06/2024)

Laura A. King (July 2001). The Health Benefits of Writing about Life Goals. (25/06/2024)

Fran Whitaker. 4 Reasons It’s Hard to Stick to a Journaling Routine. (03/08/2024)

6 Appendix

6.1 Project Proposal

project_proposal.docx

6.2 Project Plan

project_analysis_and_design_documentation.docx

6.3 Requirement Specification

project_requirements_documentation.docx

6.4 Monthly Journal

August 2024

- August 10, 2024
  - Finished conclusion section and added a video recording.
  - Tried tests but with no result.
  - Finished the GUI section.
- August 9, 2024
  - Explained the Home.js page with handleNotification as an interesting code snippet.
- August 8, 2024
  - Tried tests but with no result.
  - Finished the GUI section.
  - Planned to implement the mood chart if time permits.
- August 7, 2024
  - Deleted FeedbackSubmitted.js and removed some comments.
  - Attempted to control the edit section, but with no result.
- August 6, 2024
  - Added border change for email, password, and repeat password fields.
  - Updated screenshots for the user registration algorithm.
- August 5, 2024
  - Updated home to get alerts instead of other components.
  - Moved onto Design and Architecture.
  - Added dropdown radio stations with pause when switching and logout option.

- •August 4, 2024
    - •Still needed to complete the Settings use case.
    - •Needed to add a logout button in the settings.
- •August 3, 2024
    - •Added documents.
    - •Learned how to find URLs for radios.
    - •Added radio to the new entry section.

July 2024

- •July 31, 2024
    - •Finished code, started documentation and testing.
    - •Adjusted the color of the stars.
    - •Added alert message for login and deleted index.html in the backend.
- •July 30, 2024
    - •Updated some comments.
    - •Fixed Quotes.js.
    - •Completed dark mode, needed to redefine the color for the quotes author.
- •July 29, 2024
    - •Modified the order of dashboard and new entry buttons.
    - •Tried to publish on GitHub Pages.
    - •Added Dashboard and Journal buttons for Entry Submitted.
    - •Updated name for Journal and added dashboard button.
    - •Tried to fix a bug for editTitle when canceling and it remains in edit mode.
    - •Implemented the mood option.
    - •Mood display working for the moment.
    - •Added modal to journal entries when not in view mode.
- •July 28, 2024
    - •Satisfied with the design, only needed to add mood emoji.
    - •Decided to leave the design as it is for now.
- •July 27, 2024
    - •Added loop of 20 to test multiple entries.
    - •Finished delete logic.
    - •Added alert for deletion.
    - •Found the right balance for the logic of the delete mode.
- •July 24, 2024
    - •Fixed feedback submit address.
    - •Fixed update entry for the content.
    - •Tried to implement edit mode in entry.
    - •Managed to get backend to work to get entries by email.
    - •Finished commenting on existing files.
- •July 23, 2024
    - •Finished comments up until Quotes.js.
    - •Removed .css for some files, updated comments.

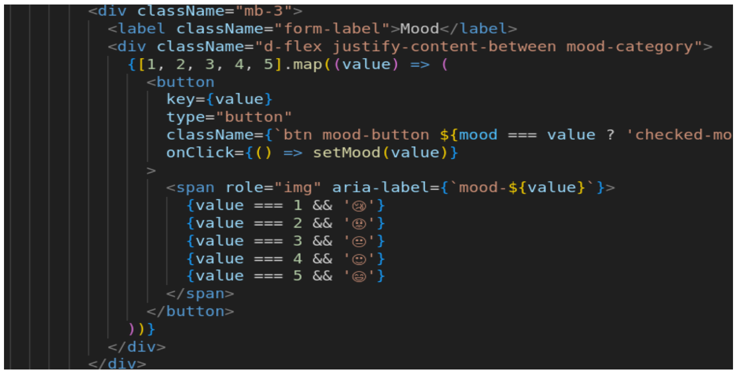
- •July 22, 2024
  - •Added comments to backend and start of frontend.
  - •Commented models.
  - •Modified names of files for better readability.
  - •Fixed registrationsuccess page.
- •July 21, 2024
  - •Stuck on entry effects.
  - •Fixed glitch visual bug for quotes by using translateY(50px) instead of -50px.
  - •Quick touch to feedback with the styling.
  - •Fixed feedback submit.
  - •Added discard option for feedback.
  - •Added discard option for create entry.
- •July 20, 2024
  - •Tried to change background.
  - •Made black and white emojis.
  - •New entry happy.
  - •Added bootstrap, added mood to new entries.
- •July 17, 2024
  - •Modified quotes.js, added previous quote.
- •July 15, 2024
  - •Ensured the endpoints for CRUD are working.
  - •Fixed link between backend and frontend for new entry.
  - •Fixed journal entry endpoint and routes to it.
  - •Fixed feedback post and get.
  - •Managed to fix the endpoints, post/get works, now needed to fix tags.
- •July 14, 2024
  - •Managed to do login logic, needed to refine Login.js page.
  - •Got stuck on the feedback post and get.
  - •Name recovery done, needed to go online with server and DB.
  - •Managed login.js route and dashboard route.
  - •Needed to refine login page.
- •July 13, 2024
  - •Issue with the login auth.
  - •Tried login auth.
  - •Added backend and frontend Registration link.
  - •Fixed screen size issue.
- •July 12, 2024
  - •Happy with the new-entry page.
  - •Finished submitfeedback and started NewEntry.css.
- •July 11, 2024
  - •Needed to refine submitfeedback.js and CSS.
  - •Finished quotes.js and dashboard.js.
- •July 10, 2024
  - •Started feedback page.

        •Added login, register page.
- •July 9, 2024
    - •Added touch to the line on top of Soul Journal.
    - •Finished home page.
- •July 7, 2024
    - •Finished server-side logic for CRUD and user.
    - •First commit.

6.5 *Other Material Used*

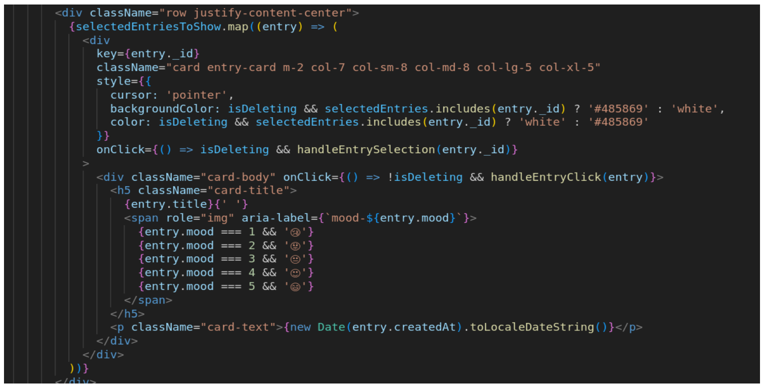
Use of **ChatGPT**

I've used **ChatGPT** to implement some functions for my web application. I will share below the screenshots with the code that was constructed with the help of ChatGPT. Even though at first I didn't know much about the function, understanding it at a minimal level, now I consider myself capable of understanding it.



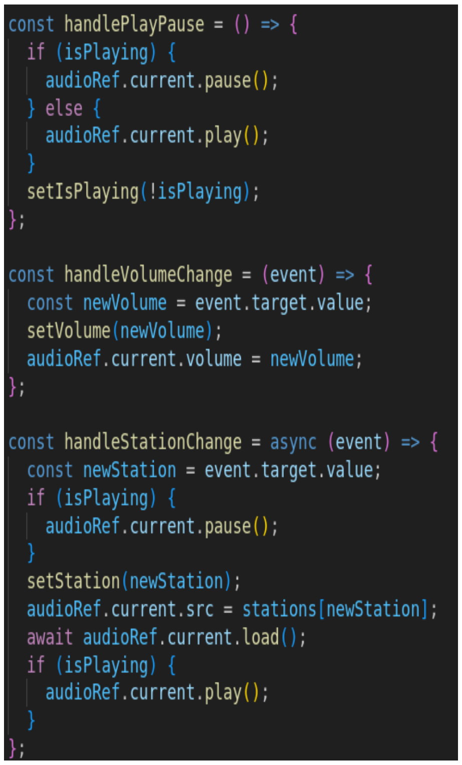
```
<div className="mb-3">
  <label className="form-label">Mood</label>
  <div className="d-flex justify-content-between mood-category">
    {[1, 2, 3, 4, 5].map((value) => (
      <button
        key={value}
        type="button"
        className={`btn mood-button ${mood === value ? 'checked-mo
        onClick={() => setMood(value)}
      >
        <span role="img" aria-label={`mood-${value}`}>
          {value === 1 && '😢'}
          {value === 2 && '😟'}
          {value === 3 && '😐'}
          {value === 4 && '🙂'}
          {value === 5 && '😄'}
        </span>
      </button>
    ))}
  </div>
</div>
```

Figure 6.5.1: Mood Selection



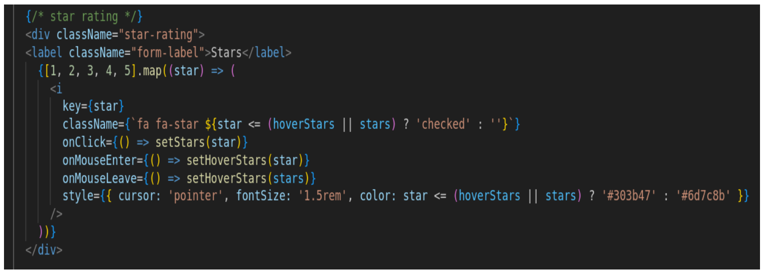
```
<div className="row justify-content-center">
  {selectedEntriesToShow.map((entry) => (
    <div
      key={entry._id}
      className="card entry-card m-2 col-7 col-sm-8 col-md-8 col-lg-5 col-xl-5"
      style={{
        cursor: 'pointer',
        backgroundColor: isDeleting && selectedEntries.includes(entry._id) ? '#485869' : 'white',
        color: isDeleting && selectedEntries.includes(entry._id) ? 'white' : '#485869'
      }}
      onClick={() => isDeleting && handleEntrySelection(entry._id)}
    >
      <div className="card-body" onClick={() => !isDeleting && handleEntryClick(entry)}>
        <h5 className="card-title">
          {entry.title}{' '}
          <span role="img" aria-label={`mood-${entry.mood}`}>
            {entry.mood === 1 && '😢'}
            {entry.mood === 2 && '😟'}
            {entry.mood === 3 && '😐'}
            {entry.mood === 4 && '🙂'}
            {entry.mood === 5 && '😄'}
          </span>
        </h5>
        <p className="card-text">{new Date(entry.createdAt).toLocaleDateString()}</p>
      </div>
    </div>
  ))}
</div>
```

Figure 6.5.2: Entries card



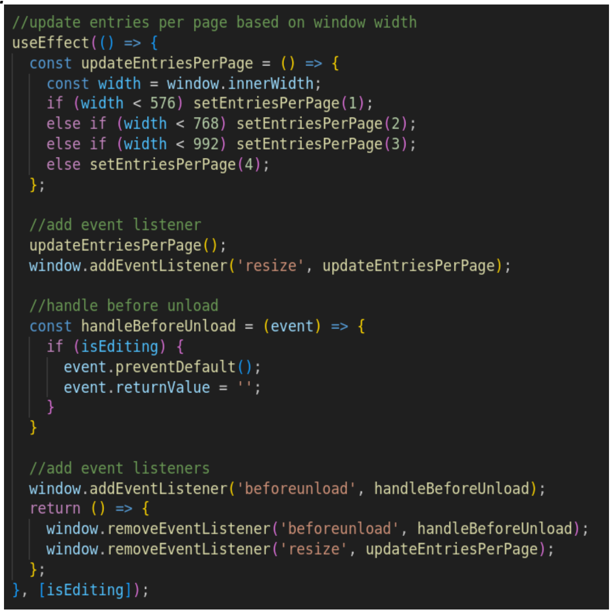
```
const handlePlayPause = () => {
  if (isPlaying) {
    audioRef.current.pause();
  } else {
    audioRef.current.play();
  }
  setIsPlaying(!isPlaying);
};

const handleVolumeChange = (event) => {
  const newVolume = event.target.value;
  setVolume(newVolume);
  audioRef.current.volume = newVolume;
};

const handleStationChange = async (event) => {
  const newStation = event.target.value;
  if (isPlaying) {
    audioRef.current.pause();
  }
  setStation(newStation);
  audioRef.current.src = stations[newStation];
  await audioRef.current.load();
  if (isPlaying) {
    audioRef.current.play();
  }
};
```

Figure 6.5.3: RadioPlayer Controls



```
{/* star rating */}
<div className="star-rating">
<label className="form-label">Stars</label>
  {[1, 2, 3, 4, 5].map((star) => (
    <i
      key={star}
      className={`fa fa-star ${star <= (hoverStars || stars) ? 'checked' : ''}`}
      onClick={() => setStars(star)}
      onMouseEnter={() => setHoverStars(star)}
      onMouseLeave={() => setHoverStars(stars)}
      style={{ cursor: 'pointer', fontSize: '1.5rem', color: star <= (hoverStars || stars) ? '#303b47' : '#6d7c8b' }}
    />
  ))}
</div>
```

Figure 6.5.



```
//update entries per page based on window width
useEffect(() => {
  const updateEntriesPerPage = () => {
    const width = window.innerWidth;
    if (width < 576) setEntriesPerPage(1);
    else if (width < 768) setEntriesPerPage(2);
    else if (width < 992) setEntriesPerPage(3);
    else setEntriesPerPage(4);
  };

  //add event listener
  updateEntriesPerPage();
  window.addEventListener('resize', updateEntriesPerPage);

  //handle before unload
  const handleBeforeUnload = (event) => {
    if (isEditing) {
      event.preventDefault();
      event.returnValue = '';
    }
  }

  //add event listeners
  window.addEventListener('beforeunload', handleBeforeUnload);
  return () => {
    window.removeEventListener('beforeunload', handleBeforeUnload);
    window.removeEventListener('resize', updateEntriesPerPage);
  };
}, [isEditing]);
```

Figure 6.5.5: updateEntriesPerPage