***SoulJournal***

Product Design Specification Template

Version *<1.0>*

*<07/17/2024>*

VERSION HISTORY

[Provide information on how the development and distribution of the **Product Design Specification Template**, up to the final point of approval, was controlled and tracked. Use the table below to provide the version number, the author implementing the version, the date of the version, the name of the person approving the version, the date that particular version was approved, and a brief description of the reason for creating the revised version.]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version #** | **Implemented**  **By** | **Revision**  **Date** | **Approved**  **By** | **Approval**  **Date** | **Reason** |
| 1.0 | *Alexandru Georgescu* | *07/17/2024* | *<name>* | *<mm/dd/yy>* | Initial Draft |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**UP Template Version:** 12/31/07

TABLE OF CONTENTS

[1 Introduction 4](#__RefHeading___Toc1695_3945709058)

[1.1 Purpose of The Product Design Specification Template Document 4](#__RefHeading___Toc1697_3945709058)

[2 General Overview and Design Guidelines/Approach 4](#__RefHeading___Toc1699_3945709058)

[2.1 Assumptions / Constraints / Standards 4](#__RefHeading___Toc1701_3945709058)

[3 Architecture Design 5](#__RefHeading___Toc1703_3945709058)

[3.1 Logical View 7](#__RefHeading___Toc1705_3945709058)

[3.2 Hardware Architecture 9](#__RefHeading___Toc1707_3945709058)

[3.3 Software Architecture 10](#__RefHeading___Toc1709_3945709058)

[3.3.1 Server-side 10](#__RefHeading___Toc1885_737508667)

[NPM Packages used: 10](#__RefHeading___Toc1887_737508667)

[3.3.2 Client-side 10](#__RefHeading___Toc1889_737508667)

[3.4 Security Architecture 11](#__RefHeading___Toc1711_3945709058)

[3.5 Communication Architecture 11](#__RefHeading___Toc1713_3945709058)

[3.6 Performance 12](#__RefHeading___Toc1715_3945709058)

[4 System Design 14](#__RefHeading___Toc1717_3945709058)

[4.1 Use-Cases 14](#__RefHeading___Toc1719_3945709058)

[4.2 Database Design 14](#__RefHeading___Toc1721_3945709058)

[4.3 Data Conversions 16](#__RefHeading___Toc1723_3945709058)

[4.4 Application Program Interfaces 16](#__RefHeading___Toc1725_3945709058)

[4.5 User Interface Design 17](#__RefHeading___Toc1727_3945709058)

[4.6 Performance 18](#__RefHeading___Toc1729_3945709058)

[19](#__RefHeading___Toc1733_3945709058)

# Introduction

## Purpose of The Product Design Specification Template Document

**Introduction**

This Product Design Specification document details the system architecture and overall system design of the SoulJournal web application. It is produced during the planning phase of application development and aims to provide comprehensive guidance on constructing the system architecture for the development team, project manager, and project team.

**Purpose**

The purpose of this document is to outline the architecture and design of the SoulJournal system to ensure a clear and structured development process. It serves as a roadmap for developers, project managers, and other project team members, facilitating a shared understanding of the system’s components and their interactions.

# General Overview and Design Guidelines/Approach

This section describes the principles and strategies to be used as guidelines when designing and implementing the system.

## Assumptions / Constraints / Standards

The design of SoulJournal focuses on being user-accessible and easy to use on all devices. We need to make sure it works well with modern web browsers and meets accessibility standars to be usable by everyone.

Security is also important, so we’ll use data encryption and secure login methods. The design will be simple and clean, making it easy to navigate. The system will be built to handle more users over time and allow for future updates. We will follow best practices in responsive web design to ensure it looks and works well on different screen sizes.

For the full functionality to be available for the user, a strong internet connection must be present.

The application will also be tested on the Firefox browser on Linux, and on the Chrome browser on Linux.

# Architecture Design

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:

* **frontend**
* **backend**
* **database**

**Components:**

1. **Frontend(Client-Side)**
   * **Technologies:** React.js, HTML, CSS
   * **Role**: provides the user interface for journaling and displaying quotes.
   * **Interaction**: Sends user inputs and requests to the backend via HTTP
2. **Backend(Server-Side)**
   * **Technologies:** Node.js, Express.js
   * **Role**: Handles business logic, processes 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 inspirational 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.
     + Ensures data integrity and security through encryption and proper indexing.
     + Returns the requested data to the backend for further processing or direct user interaction.

**Data Flow:**

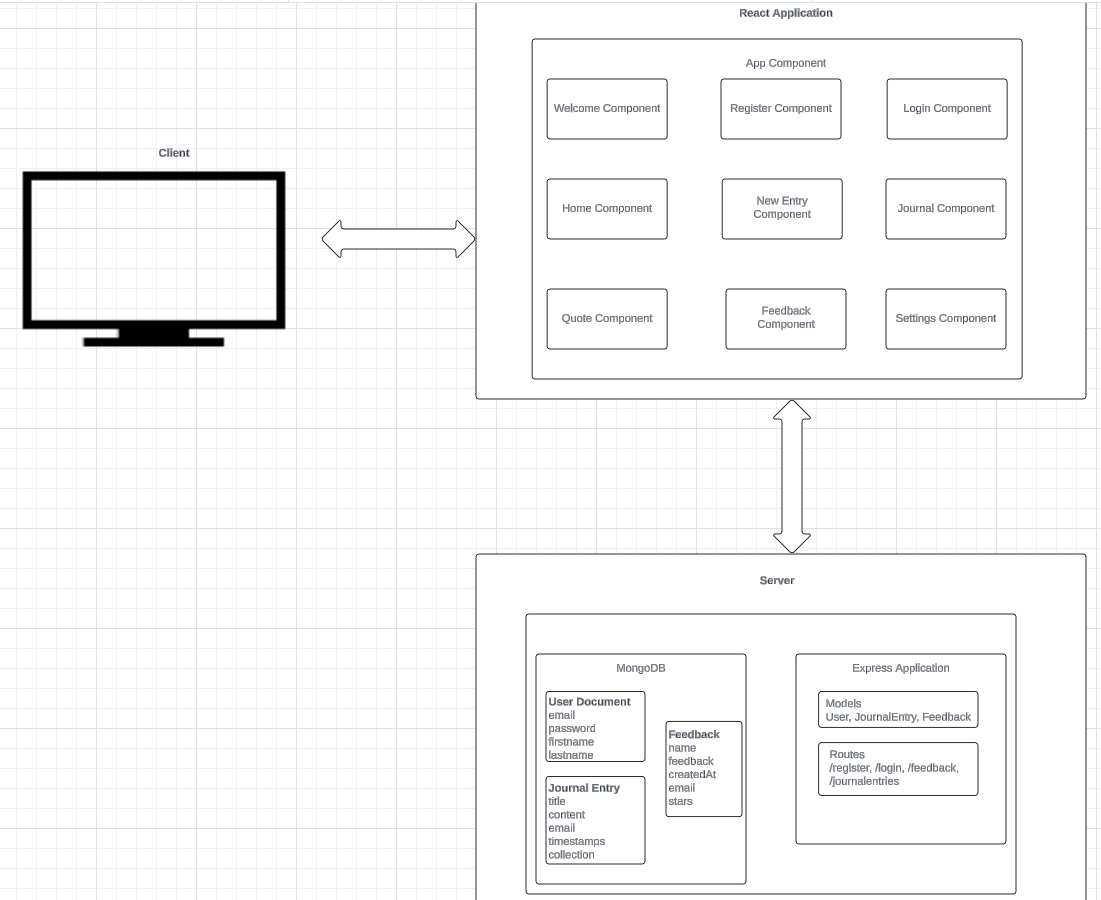
1. **User Interaction**
   * A user interacts with the SoulJournal app via the frontend.
   * Example: A user writes a new journal entry and submits it.
2. **Frontend to Backend Communication**
   * The frontend captures the journal entry data and sends it to the backend via an API call.
   * Data is sent in JSON format over HTTPS to ensure security.
3. Backend Processing
   * The backend receives the journal entry data, processes it, and prepares it for storage.
   * If an inspirational quote is needed, the backend makes a request to the Quotable API.
4. External API Interaction
   * The backend sends a request to the Quotable API and receives a quote in response.
   * The received quote is processed and stored or directly passed to the frontend.
5. Database Interaction
   * The backend sends the journal entry data to the database for storage.
   * The database stores the data securely and confirms the operation back to the backend.
6. Response to Frontend
   * The backend sends a confirmation response to the frontend, indicating the success or failure of the journal entry submission.
   * The frontend updates the user interface based on the response, displaying confirmation messages or errors.

#### External Interactions

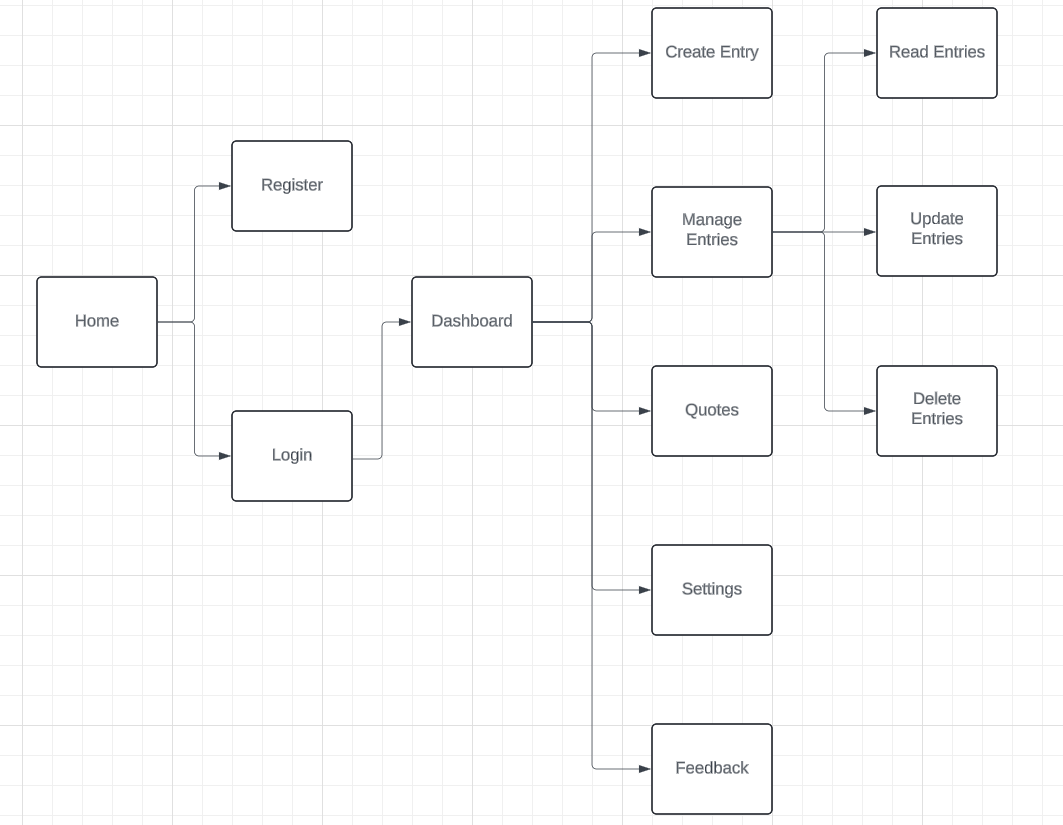
* Quotable API
  + The backend interacts with the Quotable API to fetch inspirational quotes based on user interactions.
  + Ensures that data exchange is secure and the API responses are appropriately integrated into the user experience.

## Logical View

#### Below you can see the Logical view for the **SoulJournal** architecture. (Fig 3.1.)

Figure 3.1 Logical View Diagram

In the representation from **Figure 3.1**, the client is basically a user’s browser on any device, such as a tablet, phone or even a desktop. When a user visits the SoulJournal web application’s URL, the React application loads in their browser and displays the app interface. Users can also navigate through different “pages” within the app, managed by React Router.

Figure 3.2 Site Map

The above diagram (**Figure 3.2**) represents a simple site map for SoulJournal. Most navigation revolves around creating, viewing and maanaging journal entries and quotes. The key navigation paths are as follows:

1. **Home Page**
   * Displays an overview of the application
   * Users can navigate to login or register from here
2. **Login/Register**
   * Users can log in to their existing accounts or register for a new account
   * After logging in, users are redirected to their journal dashboard.
3. **Journal Dashboard**
   * Displays the menu for Journal Entry Creation, Journal Entry Viewing, Settings and Feedback.
4. **Journal Entry Creation**
   * Users can create a new journal entry
5. **Journal Entry Viewing**
   * Users can view their past journal entries, including the date, and content.
   * Entries can be edited or deleted from this page.
6. **Settings**
   * Users can manage their account settings and preferences.
7. **Quotes**
   * Users can view motivational quotes and also select to see the previous ones.
8. **Feedback Page**
   * Users can submit feedback about the application.
   * Feedback can include ratings, comments or suggestions for improvement

**User Navigation and Interactions**

1. **Navigating the App**
   * When a user visits the SoulJournal URL, the React application loads in their browser.
   * Users can log in or register, browse their journal entries, and access their profile and settings.
2. **Creating and Viewing Journal Entries**
   * From the dashboard, users can create new entries by clicking the “New Entry” button.
   * Users can view existing entries by selecting them from the list.
   * Each entry displays the date, mood, content, and any associated inspirational quotes.
3. **Settings**
   * Users can access their profile to update personal information and account settings.
4. **Providing Feedback**
   * Users can navigate to the feedback page to offer comments and suggestions
   * The feedback page allows users to rate the app and submit detailed feedback to help improve the application

## Hardware Architecture

At the moment, all the clientside, serverside and database are stored on my laptop. The specifications of the laptop are as follows:

* **Model**: HP 240 G6
* **Core:** Intel I3 7th Generation
* **RAM:** 16GB
* **Operating System:** Linux LUbuntu

## Software Architecture

A visual overview of the system architecture can be seen in figure 3.1.1.(Client-side) and figure 3.1.2(Server-side)

The application itself will be developed with the following technologies.

### Server-side

|  |  |
| --- | --- |
| **Node.js** | A server-side version of the JavaScript Language. |
| **Express.js** | A web application framework used to build RESTful APIs |
| **MongoDB** | A NoSQL database |
| **Mongoose** | An ODM (object document mapper), used for managing the interaction between the server and the database |

### **NPM Packages used:**

|  |  |
| --- | --- |
| **axios** | A promise-based HTTP client for the browser and Node.js, used for making HTTP requests. |
| **bcrypt** | A library to help with hashing the passwords. |
| **cors** | A package for providing a Connect/Express middleware that can be used to enable CORS (Cross,Origin Resource Sharing) with various options. |
| **dotenv** | A module that loads environment variables from a .env file into process.env |
| **Express** | A fast, minimalist web framework for Node.js |
| **Jsonwebtoken** | A library to sign, verify, and decode JSON Web Tokens. |
| **Mongoose** | An ODM (Object Data Modeling) library for MongoDB and Node.js, providing a straightforward, schema-based solution to model your application data. |

### Client-side

|  |  |
| --- | --- |
| **Axios** | A promise-based HTTP client for the browser and Node.js, used for making HTTP requests. |
| **React** | A JavaScript library for building user interfaces, particularly single-page applications where you can create reusable UI components |
| **React-Dom** | A package that provides DOM-specific methods that can be used at the top level of a web app to enable an efficient way of managing DOM elements of the web page. |
| **React-Router-Dom** | A collection of navigational components that compose declaratively with the application, used for routing in React applications |
| **React-Scripts** | A set of scripts from Create React App to build, test, and run your React application. |

## Security Architecture

**User Authentication and Authorisation**

When a user logs in to the website, the server-side application will create a JSON Web Token (JWT). This token acts as an identifier, which the client-side application will attach to each request sent to the back-end. This allows the server to recognize that the current user is logged in.

The JWT will be included in the Authorization header of each subsequent request, enabling the user to access routes, services and resources that are permitted with that token. This mechanism ensures that the application knows which routes the user can access and what functionality they have access to.

## Communication Architecture

The SoulJournal web application involves several types of communication between the front-end and back-end, as well as within each layer.

**Front-End to Back-End Communication**

The front-end application communicates with the back-end application via AJAX requests. When a user submits form data or credentials, these are sent to the backend. For logged-in users, additional headers are included in the form of an Authorization header containing the user’s JSON Web Token (JWT) for authorization

The server responds to the client in two ways:

1. **Static Assets:** HTML, CSS, and JavaScript files needed to load the application
2. **Dynamic Data:** json data in response to the client requests.

**Front End Internal Communication**

Within the frontend, information is communicated using “props” and “state”.

* **Props:** Pieces of information passed fron one React component to another
* **State:** These are used to update the cucrent application state. The state can then be accessed and used by any component across the application, with the information being accessed via props.

**Back End Internal Communication**

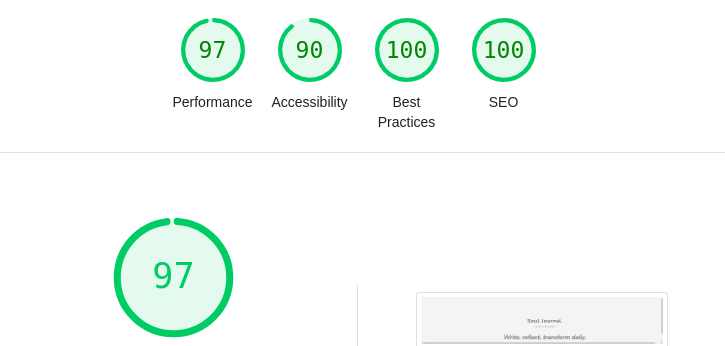
In the back-end, the Express application communicates with the MongoDB instance.

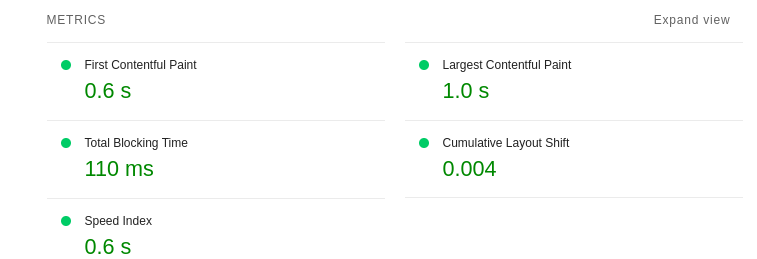
For example, the User model is represented using a Mongoose Schema. The schema is defined in a single file and exported at the end, allowing the User model to be imported into any other file in the server-side application. This enable full Mongoose functionality to interact with the User document in the MongoDB database.

## Performance

Node.js excels in handling multiple tasks simultaneously due to its single-threaded, non-blocking, and asynchronous nature. This allows it to process numerous operations concurrently within a single thread, enhancing the efficiency and responsiveness of the server.

On the client side, SoulJournal will leverage the speed of React. As users navigate through the app, React loads new pages instantly using JavaScript, without the need for repeated server requests for HTML and associated assets. This results in quick page transitions, with only occasional requests for JSON data to update the UI. This architecture ensures a smooth, fast, and responsive user experience. Below in Figure 3.6.1 and Figure 3.6.2 we can see a Lighthouse report on the web application.

Figure 3.6.1: Lighthouse Report

Figure 3.6.2: Lighthouse Metrics

# System Design

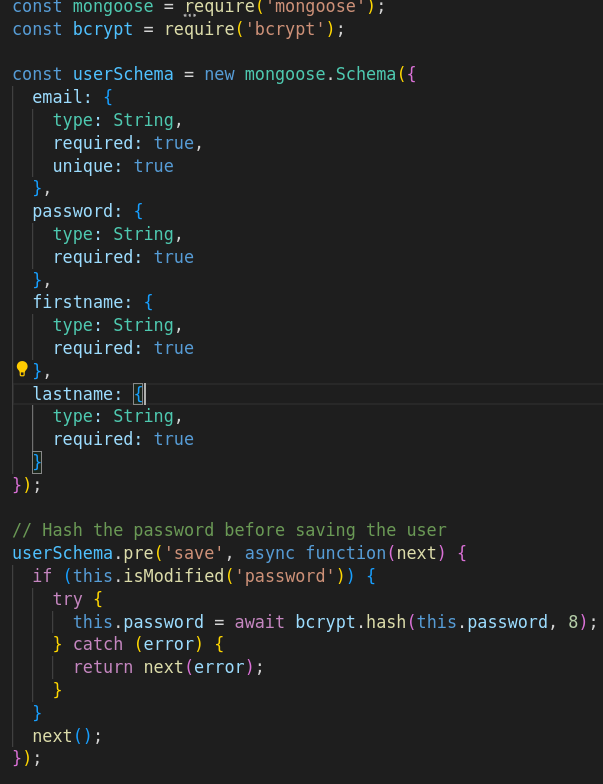
## Use-Cases

The use cases for the SoulJournal web application can be found in the Requirements Specification documentation in the section 3.

## Database Design

The SoulJournal web application utilizes a MongoDB instance hosted on the local storage. MongoDB is a NoSQL database, which means it stores information in “documents” that are structured similarly to JSOn objects. This approach aligns well with the application’s use of JavaScript, as the data format remains consistent throughout.

To manage interactions with MongoDB, the application utilizes Mongoose, a JavaScript-based Object Document Mapper(ODM). Mongoose helps in defining scehmas, making it straightforward to query and manipulate data within the database. Figure 4.2.1 demonstrates an example schema using Mongoose.

Figure 4.2.1: Mongoose User Schema and Hashing

The code above defines a Mongoose schema for a User model in a Node.js application. This schema outlines the structure of the User documents that will be stored in the MongoDB collection.

This approach ensures that user passwords are securely hashed before being stored in the database, enhancing the security of the application by protecting user credentials from being stored in plain text.

## Data Conversions

Data exchanges within the SoulJournal application will be in the form of JavaScript Object Notation(JSON). JSON is a lightweight data format that resembles JavaScript object syntax, making it easy to work with in a JavaScript-based application.

When the client requests data from the server, the server responds with JSON. This JSON data is then parsed by the client into JavaScript objects or arrays, making it ready for use in the application.

Conversely, when the client submits form data to the server, the data is first converted into JSON format. This JSON data is then sent to the server, where it is parsed into usable objects that can be manipulated and stored in the database.

For example, JSON data might look like this:

“journalEntry”: {

title: { type: String, required: true },

content: { type: String, required: true },

email: { type: String, required: true } // Move email field inside the schema definition

}

In this example, a journal entry is formatted in JSON. This format ensures that data can be easily transferred bertween the client and server, facilitating seamless interaction within the SoulJournal application.

## Application Program Interfaces

An API is more like a bridge for external communication with an application or service. For the SoulJournal web application,we will utilize the Quotable API to fetch inspirational quotes. When a user interacts with SoulJournal, the application sends an HTTP request to the Quotable API, which then retrieves a relevant quote and responds with the information in JSON format. This allows SoulJournal to provide users with dynamic and timely inspirational quotes based on their activities within the app.

Additionally, SoulJournal employs MongoDB to store journal entries and user data. Communication with the MongoDB database occurs through secure connections, with data being exchanged in JSON format. This ensures seamless integration and data manipulation within the JavaScript-based environment of the application.

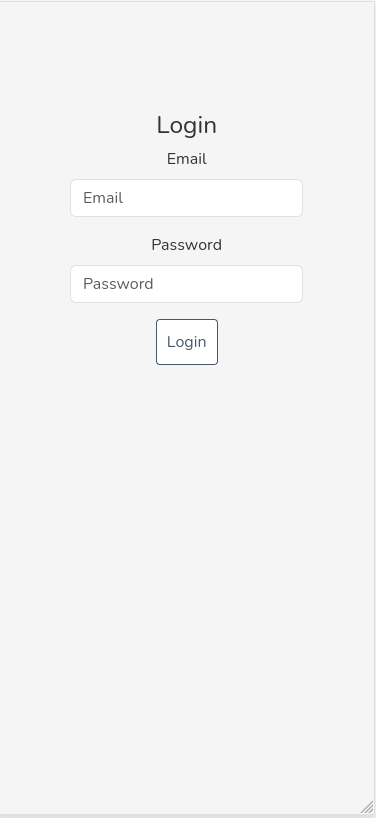
## User Interface Design

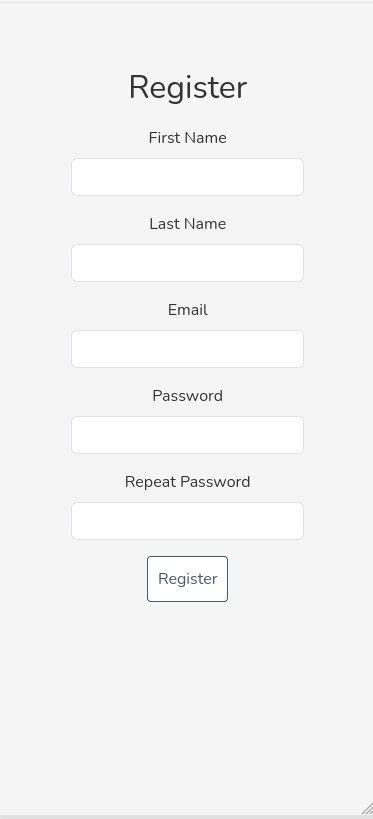
SoulJournal will have an interface made using the following technologies:

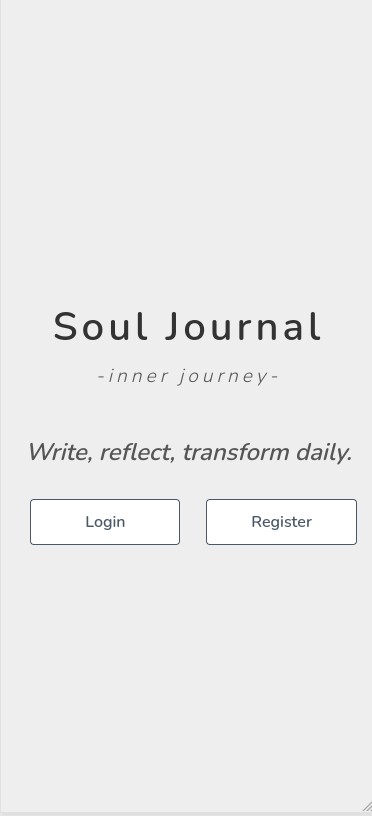
For the visual design, we will utilize BootStrap. This technology is a widely-used CSS framework that enables rapid website development with pre-built styles and components. Its grid-based system ensures that the application is mobile-responsive, significantly reducing development time and effort.

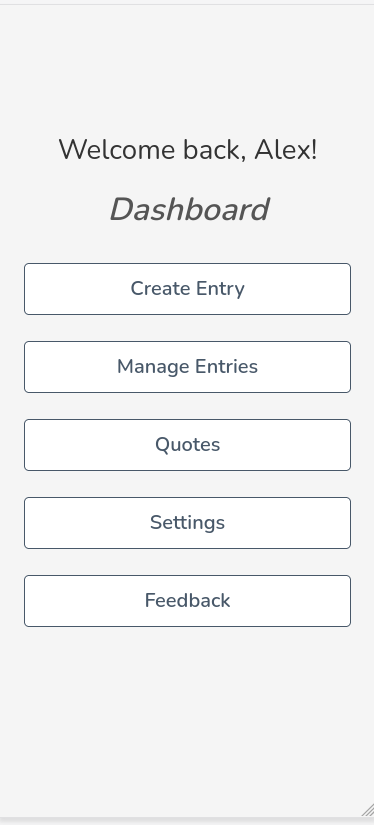
The interactive functionality of the user interface will be managed using React. React is a client-side JavaScript library designed to build dynamic and responsive web applications. It allows us to create reusable UI components, ensuring a smooth and interactive user experience.

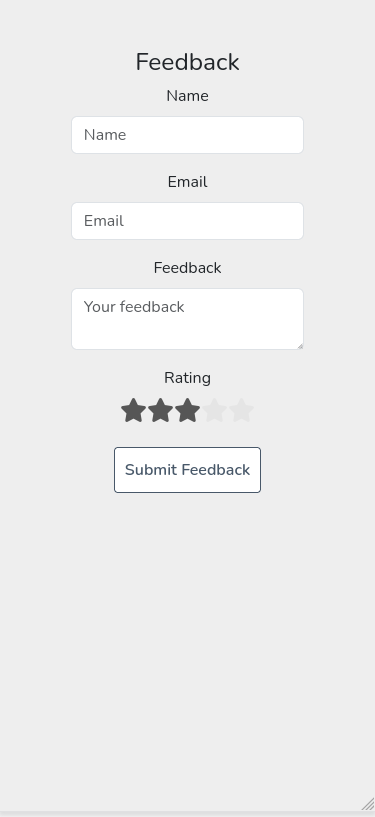
Below in the following figures (Figure 4.5.1-4.5.6) I will display the current state of the web application:

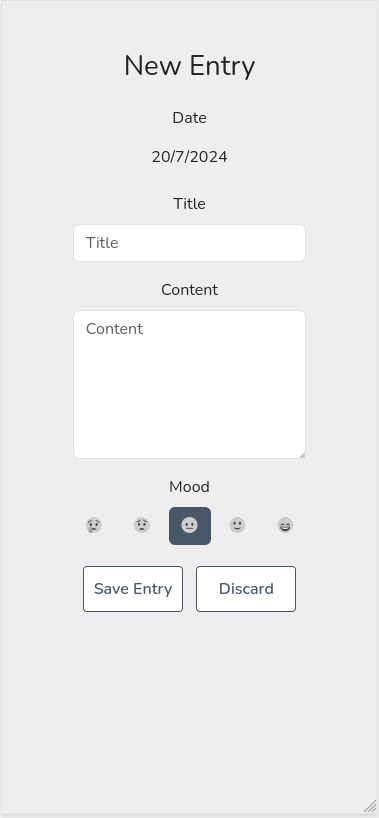
4.5.3: Login Page

4.5.2: Register Page

Figure 4.5.1 / Page

4.5.4: Dashboard Page

4.5.5: Feedback Page

4.5.6: Entry Page

# 

Appendix A: References

The following table summarizes the documents referenced in this document.

|  |  |  |
| --- | --- | --- |
| **Document Name and Version** | **Description** | **Location** |
| Requirements Specification | *Description of the Use Cases for the user* | *Local Storage* |

Appendix B: Key Terms

The following table provides definitions for terms relevant to this document.

|  |  |
| --- | --- |
| **Term** | **Definition** |
| *API(Application Programming Interface)* | *A set of rules and tools that allow different software applications to communicate with each other* |
| *CSS(Cascading Style Sheets)* | *A style sheet language used for describing the presentation of a document written in HTML.* |
| *HTML(HyperText Markup Language)* | *The standard language used to create and design documents on the World Wide Web* |
| *JWT(JSON Web Token)* | *A compact, URL-safe means of representing claims to be transferred between two parties, typically used for authentication and authorization.* |
| *JSON(JavaScript Object Notation)* | *A lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate.* |
| *Node.js* | *A runtime environment that allows you to execute JavaScript code on the server-side.* |
| *ODM(Object Document Mapper)* | *A programming technique for converting data between incompatible type systems in object-oriented programming languages. Mongoose is an example of an ODM for MongoDB.* |
| *REST(Representational State Transfer)* | *An architectural style for designing networked applications, relying on stateless, client-server communication typically using HTTP.* |
| *React.js* | *A JavaScript library for building user interfaces, especially single-page applications where you can create reusable UI components* |
| *HTTP(HyperText Transfer Protocol)* | *An application protocol used for transmitting hypermedia documents, such as HTML, across the Internet.* |
| *NoSQL(Not Only SQL)* | *A type of database that provides a mechanism for storage and retrieval of data modeled in means other than the tabular relations used in relational databases.* |
| *Bcrypt* | *A password hashing function designed to secure passwords by adding a salt to protect against brute-force attacks.* |
| *Express.js* | *A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.* |
| *NPM(Node Package Manager)* | *A package manager for JavaScript, used to install, share, and manage code modules written in JavaScript.* |
| *CORS(Cross-Origin Resource Sharing):* | *A mechanism that uses additional HTTP headers to allow a web application running at one origin to access resources from a server at a different origin.* |