Computing NEA - Journal App

Xichao Wang

March 27, 2024

Contents

Chapter 1

Abstract

This project describes the development of a full-stack journal application. The app encompasses a Django Server controlling a PostgreSQL database, a React frontend that communicates with the server through RESTful APIs exposed by the Django server.

Chapter 2

Analysis

2.1 Problem Area

The issue identified is the lack of mindfulness and increased forgetfulness. Keeping a journal acts as a way to document significant moments within an individual's life while providing a personal and private way for people to express their emotions. I find myself a perfect example of somebody that is lost in 'modern life,' leading me to forget even simple events which had occurred within my day.

In a hectic world full of day-to-day distractions, the creation of a consumerist culture through the rise of social media has increased the likelihood for individuals to feel less content with their day-to-day lives. We are consuming content through different forms of media, whether that be through our computers, phones or televisions. As a result, individuals are left constantly craving to ingrain more information, but none of this information is retained. The society of the modern world has evolved to the extent that it has made it inevitable for individuals to be exposed to the high-volume of content. Such content is easily accessible through our phones, inevitably leading individuals to become overwhelmed and overloaded. If we consider the Covid-19 quarantine and its long-lasting impact on teenagers' mental

health, [5] it has left a void for individuals needing to relieve their anxieties and reduce the overall stress incorporated into their daily lives. journaling can induce a positive impact on individuals struggling with the overwhelming nature of social media and how it has clearly merged itself with day-to-day life, bringing mindfulness as a positive method in documenting and organising a person's life.

Keeping a Journal increases productivity and mindfulness. I have always wanted to incorporate this habit into my daily routine, and through my NEA, I hope to create an easily accessible platform for myself and those alike. For my NEA, I want to create a minimal, easily navigated web-interface where individuals can add entries, storing the data, and through my web-interface, these entries can be safely stored for personal use for whoever is accessing the website.

2.1.1 Benefits of journaling

Here are some benefits of journaling include:

- It is a method of mindful writing that can help you to become more aware of your thoughts and feelings. Therefore come to terms with them.
- Simply jot down your thoughts and ideas, and you can come back to them later.
- It can help you to become more organised and productive by making you more aware of your goals and aspirations.

2.2 Client / End User

journaling is an excellent habit anyone could incorporate into their daily lives. However, my primary target end-user for my app would be people like

myself who are more digitally orientated and are comfortable with accessing an online platform to manage their daily lives. Teenagers and young adults are more likely to be my target audience, as they are more likely to be comfortable with using online platforms.

I've wanted to journal for quite some time now, but due to my busy schedule and overall forgetfulness, I have never been consistent with the habit for over a week. To finally properly start journaling once and for all, I will build a website front end that can be accessed as long I have an internet-connected device. This way, I can journal no matter where I am, reducing the friction that prevents me from cultivating the habit of journaling.

2.2.1 Survey Result

I have created a survey for friends and family to fill out to understand the needs and people's thoughts about journaling. Through this, I learned about more benefits of the habit and I also gained insight into people's online usage which will help me to design the app. Here are some of the results from the survey:

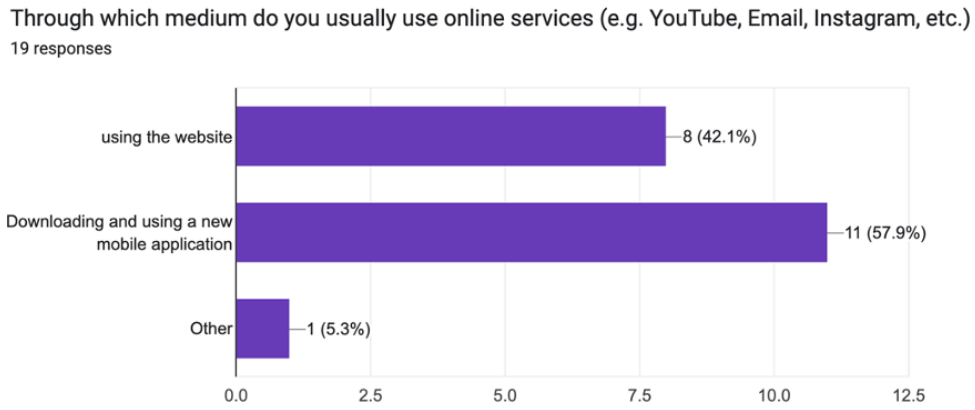


Figure 2.1: Survey Results Showing the Platform People Use to Access Online Services

The figure above shows that although most people use mobile applications to access online services, a significant number of people use website. I will be creating a website for the journal app because it's universally accessible, but in the future, I can create a mobile app to cater to the needs of the users.

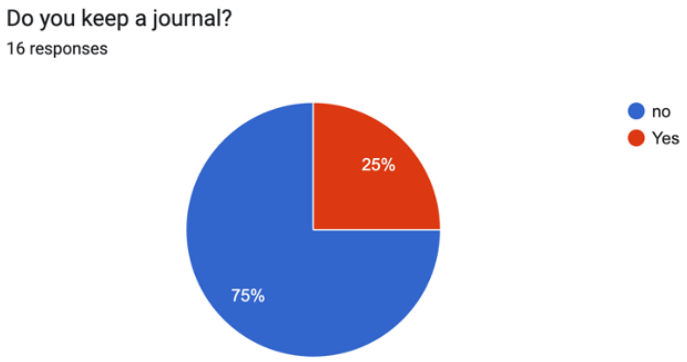


Figure 2.2: Survey Results Showing the Ratio of People who Keep a Journal

As expected, the majority of people do not keep a journal. Creating my app could potentially help people to start journaling and experience the benefits of the habit.

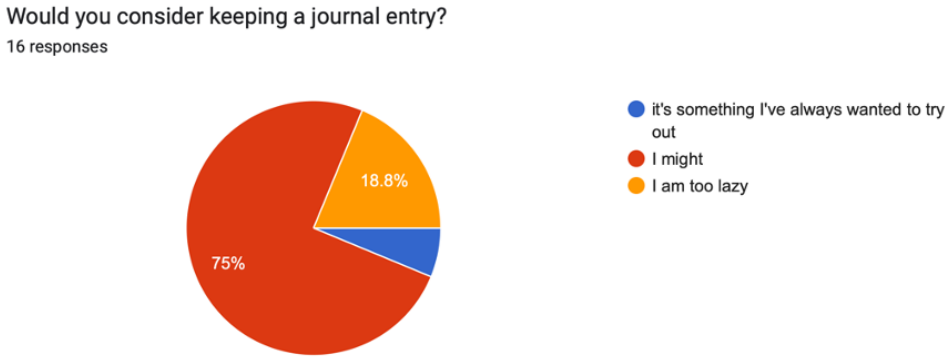


Figure 2.3: Survey Results Showing the Ratio of People who Would Consider Journaling

Interestingly, a significant number of people would consider journaling. This shows that there is a demand which I could potentially fill with my app.

Q1. *I asked the participants who journal what inspired them to start journaling.*

A1. "I overthink too much"

"Was more of a note taking thing I used to do then it evolved into writing - not all the time, but on occasion"

"Having a busy period in my life where so much was going on that I realised that I wanted to remember as much of it as possible. However, I wasn't certain that I'd be able to recall everything on my own, so I wrote things down as they happened to ensure that I wouldn't forget them."

"I was walking across the river Thames one day and I sat down for 30 minutes and just stared as the river flowed extremely peacefully. I found so many thoughts racing through my head and got restless without any digital media to entertain me. This experience inspired me to have that peace of mind and articulate my thoughts."

- This confirms my belief that journaling is a good habit with many benefits.

Q2. *I then asked everyone for the benefits they see in journaling.*

A2. Results

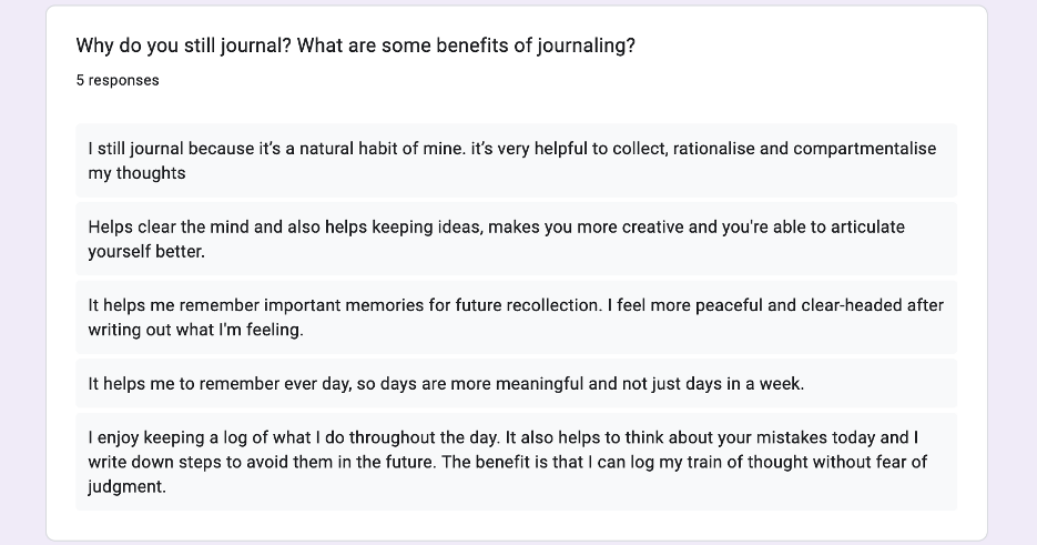


Figure 2.4: Survey Results Showing the Benefits of Journaling Highlighted by People Who Journal

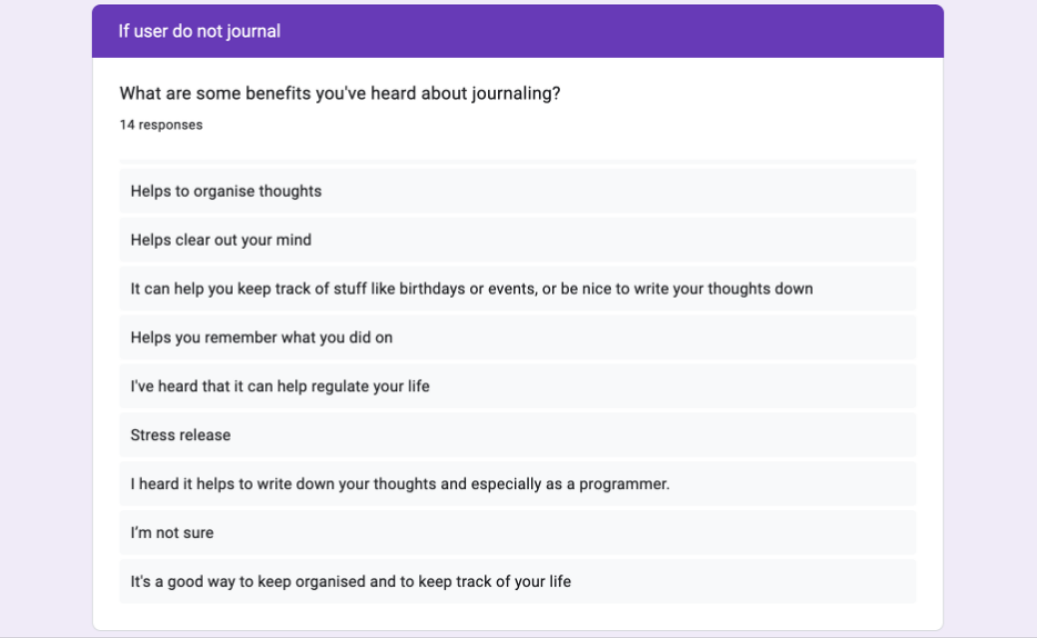


Figure 2.5: Survey Results Showing the Benefits of Journaling Highlighted by People Who Do Not Journal

Regardless of whether people journal or not, they all see the benefits of journaling. Almost every single person has a unanimous agreement that journaling is a good habit to have. This is a good sign for my app as I was correct for thinking that people would be interested in journaling.

Q3. *Finally, I asked the participants "What would be some important and fundamental features of a digital journal app?"* Here are a few selected responses that represent all the responses as a whole:

A3. "Security, I don't want my data to get breached and leaked."

- This is a very important feature to have in the app. I will need to ensure that the data is stored securely and that sensitive information is encrypted.

A3. "Writing entries, being able to select and view a past date, low-key it would be really interesting to see my journaling pattern in the long term, if you can do that itd be kinda cold"

- The thing to highlight here is that the user wants to be able to refer back to past entries and be able to see the statistics of their journaling. This is a good idea and I will need to implement this feature in the app.

Interview with Miss Milanova

I had an interview with a teacher at my school who represents someone who's slightly older than me and represents the wider user base of the app. I have picked out some responses which are particularly relevant to the application.

Q1. *How do you feel about the security of your personal information in digital platforms? Are there expectations you have?*

A1. "...Yeah, surely my password and logins should be protected and private"

- Users of the platform want to have a way of securely storing their sensitive information. Therefore, I will need to implement a secure way of not only storing users but also encrypting the data that is stored there. For demonstrating purposes I will be encrypting the most sensitive data such as the user's password.

Q2. *Can you describe what you look for in a good app interface? Are there any design elements or navigation styles you find particularly helpful or annoying?*

A2. "...I would like to have an easy-to-use menu with options instead of a gimmicky and too-visual website."

- The user wants a simple and easy-to-use interface. Therefore, I will need to design a clean and feature-rich interface. I think a navigation bar would be a good idea to implement as it clearly shows all the options available to the user.

Q3. *How important is an app's speed and responsiveness to your overall user experience? Have you stopped using an app because it was too slow or unresponsive?*

A3. "Yes, very much it's a waste of my time, I would rather use something else."

- Performance is a key factor in the user experience. Therefore, I will need to ensure that the app is responsive and fast. I will need to use a framework that is known for its speed and performance.

Q4. *How has your experience been with digital note-taking or journaling tools compared to traditional methods?*

A4. "No one carries a pen and paper anymore, dont be silly... Yeah [digital journaling] would be very useful for people on the go."

- Website is accessible through all devices with an internet connection. Creating a website lays the foundation of the app, providing a good way for the user to do the journaling. In the future, I can add different forms of application.

Q5. *Any additional comments?*

A5. "I find it frustrating with other apps to enter the date every time I want to log something. Can't it just record it automatically Automatically - you can have your morning meeting then later you can have your evening meeting it automatically enters the date. It would be quite handy."

A5. "I like it the data sync between different devices."

- User wants to be able to have consistent data for ease of access. Building an API and Backend means I can store the data in a way that is easily accessible and consistent. These data can then be accessed through all forms of Frontend interfaces. This proves that in the future I can create other interfaces such as a mobile app and be able to have consistent data via the Backend.

2.3 Research Methodology

Initially what inspired me to create a mindful journal app was when I watched a YouTube video that introduced the concept of journaling, it highlighted the vast array of benefits that journaling can bring to an individual. Then, I looked through various online articles to understand the concept a bit more.

A reason for the creation of the app was that I wanted to create a habit of journaling, and I thought that creating an app would be a good way to do so. I have also conducted a survey and an interview to understand the needs of the user and to understand the requirements of the app.

Some of the research methods I have used include:

- Surveys
- Interviews
- Watching YouTube videos
- Online Research

- Personal Experience

2.4 Features of Proposed Solution

Below I have synthesised the core features of the proposed solution. I have broken down the features into two categories - Frontend and Backend.

2.5 Requirements Specification

ID	Requirement Description	How to Evidence
1.1	Develop a login page to authenticate users, include forms for user input and dynamically provide error/feedback when necessary.	Video of interaction with the app
1.2	Implement a registration page enabling new users to create an account, take in input for email, password, last name, and first name, and interact/update Backend. Provide feedback to the user after submission.	Demonstrate through UI screenshots or video, and code snippets.
1.3	Develop a place where the user can see details about themselves ie their personal information and journal statistics	Provide screenshots and code snippet
2.1	Create a page for users to compose new journal entries. Input parameters are the title and content of the entry. Interact with the corresponding endpoint and provide feedback to the user.	Video of the UI as well as checking database to verify successful submissions as well as code snippets.
2.2	Design a retrieve journal page that lists all the user's entries, with options for sorting the entries in ascending or descending order based on creation date or other criteria in an efficient way.	Video documenting interactions with the UI, showing the features listed.
3.1	Implement a navigation bar that adjusts its visibility of options based on the user's login status.	Video of the UI and code snippets.
4.1	Utilize state management techniques to keep the user logged in across different pages, preserving session information securely.	Screenshot and explanation of the implementation as well as video of functioning app.
4.2	Handle fetching, posting, and updating data through JSON responses from the Backend.	Screenshot of code snippets showcasing a couple of examples of this in action.

Table 2.1: Frontend Requirements for Journal App

ID	Requirement Description	How to Evidence
1.1	Design data models for users, and journal entries, with relationships defined.	Screenshot of my database admin panel with created models and their relationship as well as code snippet of the definition of the models.
1.2	Have an effective way to check the submitted user information, journal entries, and related data before storing it in the database, ensuring the data stored are in the required format.	Code snippet and testing of the functionalities.
1.3	Have a way to generate statistics of the user's journaling habits, such as the most active month of the user, average entries made per week, etc.	Code snippet of the function that generates the statistics and screenshots of the statistics being displayed.
2.1	Develop API endpoints with functionalities implemented to handle user authentication (login and registration) and other core endpoints that drive the journal app as a whole	use tools to run requests against the endpoints and capture their responses, testing different cases.
2.2	Features for secure password storage(salting and hashing) and and authenticate; token generation for session management.	Run tests to see the functionalities are working
3.1	Ensure all API endpoints are secured and accessible only to authenticated users, using tokens or similar mechanisms for session management.	screenshots of authenticated requests and their responses.
3.2	Ensure smooth and spontaneous communication with the Frontend.	Provide video of the working application
4.1	Design the Backend with room for more functionalities	Code snippets of non-core functionalities being implemented.

Table 2.2: Backend Requirements for Journal App

2.6 Critical Path

The critical path refers to the sequence of the stages in the development of my application. I am doing full stack development, which includes independent Backend and Frontend both with many small features to add all the time. The Agile methodology would be suitable for me to follow for my project. This is because it is a flexible and iterative approach that allows repeated tests of all the small modules and eventually builds up to a complete solution. It is also a good way to manage the project as I can easily adapt to changes, and I will need to make many changes since there will be many new things I will learn and obstacles overcome to as I am developing the app.

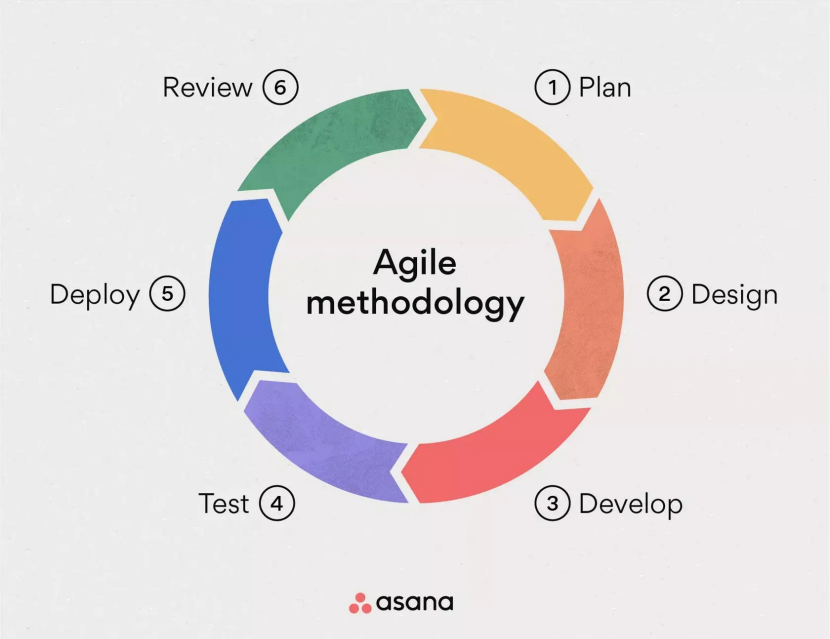


Figure 2.6: Imagine Showing the Steps of the Agile Methodology taken from [6]

Chapter 3

Design

In this section of my NEA, I will be including details on the detailed design of my Journal app.

3.1 Introduction

The objective of my journal app is to provide users with an easily accessible and navigated app for digital journaling. To accomplish this, I will choose the Django REST framework for my Backend server functionality and React, a powerful JavaScript library, for my Frontend. In this section of my report, I will be justifying why I have made these choices and detailing the specific designs for full-stack applications tailored to these frameworks

3.2 Frontend

Frontend primarily refers to the interface with which the user interacts. The most common form of Frontend is a webpage, which encompasses the layout and design elements of the website. Initially, webpages are created using

HTML, CSS, and JavaScript. However, as time progresses, developers have devised ways to generate them. Using Python, for example, frameworks like Django and Flask allow developers to build HTML, CSS, and Javascript pages interactively and dynamically. In the case of Django and Flask, developers can use templating engines like Jinja2 to generate dynamic HTML pages by integrating Python code into the HTML layout, as well as being able to avoid boilerplate code by building templates for the layout of the appearance of the page and then being able to make changes to different pages while maintaining consistency across the website. This approach saves time since it lets me focus on the features instead of writing boilerplate code.

Behind the scenes, the Frontend is not only the UI, as it also includes the logic and functionality that drive the user experience. This includes handling user interactions, making API calls, data manipulation, and updating the UI based on user input.

Javascript frameworks are the most popular amongst web developers since there exist powerful libraries like React and Angular maintained by giant tech companies, and JavaScript is simply more widely adopted and used in the web development world. Similarly, I have heard good things about a library called Svelt, which offers a lightweight approach to building user interfaces. What I mean by that is it uses fewer codes, is truly reactive, and has no virtual DOM

Initially, I used Django to generate my Frontend pages using its built-in templating engine dynamically. However, I want to create a more modular approach and, therefore, truly isolate my Backend and Frontend. The benefit of doing so is that I could connect a completely different front-end interface to the same Backend functionality with ease via API communication. In my application, I will try my best to maintain the Single Responsibility design principle throughout my application development process.
To achieve this modular approach, I decided to use a separate frontend to my Django Backend, specifically React.js. React is the most popular and in-demand ecosystem developed by Facebook for building responsive user interfaces, Facebook developed React.js and later React native for building "truly native" mobile apps using JavaScript. React.js is the javascript library I chose for my web creation, and I believe that it would be a valuable learning experience to take on my first Javascript library to build my app.

3.3 Backend

Backend refers to the server side of a web application, where the logic behind the application is. It is responsible for processing requests, interacting with databases, and algorithmically generating responses to be sent back to the client and other background processes. There are frameworks available for different programming languages that can be used for Backend development.

I have considered Python, C#, and Javascript frameworks for my project as I am proficient in Python, C# (having dabbled with those in my own time and studied those languages for GCSE and A-Levels, respectively), and Javascript is a language I have always wanted to explore further since it is the most popular language; hence, there's wide support for it in terms of libraries and frameworks. Specifically, I have considered Django, FastAPI, Flask, Next.js, and ASP.NET Core frameworks.

3.3.1 ASP.NET:

ASP.NET is a powerful framework developed by Microsoft for building web apps and services using the .NET framework and C#. It is capable and scalable, making it suitable for large-scale applications. Personally, however. I did not end up choosing ASP.NET due to my hardware constraint. I am running a MacOS unix system on the Apple silicone architecture, which I could technically do .Net development using. However, from my experience in the past, it is a pain. In addition, ASP.NET was primarily suited for the Windows ecosystem (although they are trying to make it more platform-independent), meaning it is not optimal for me personally, and there are other better ways to create a university platform-independent Backend.

3.3.2 Next.js and Express.js:

Two of the most demanding and powerful frameworks are these two are used to build the Backend using Node.js. Next.js goes hand-in-hand with React

since it provides server-side rendering. This is a thin client approach, where the initial HTML is generated on the server and then sent to the client. This approach ensures the smoothness of the user experience [4]. Express.js is, on the other hand, a lightweight and flexible framework for building RESTful APIs and the like. It might be nice, but I am more comfortable with Python, so I decided that it's a good idea to only use Javascript for my Frontend development and Python for my Backend.

3.3.3 Flask and FastAPI:

Flask is a lightweight web framework for Python used to build APIs. I had previously dabbled with the framework and found it barebone and flexible. FastAPI is a much more modern alternative to Flask. It provides similar simplicity and flexibility but with the added benefit of high performance due to its asynchronous capabilities. I was tempted to use FastAPI for my project, however, I had been learning a lot of Django and decided to go with it instead.

3.3.4 Django

Django is the most feature-rich mainstream Backend web development framework in Python. It provides all the necessary tools I need, including URL routing, Database Object-Relational mapping, as well as overall security features preventing basic forms of attacks such as CSRF. This allows me to robustly build my application, focusing on functionalities that matter rather than reinventing the wheels. Django also allows me to have fine-tuned control over functionalities if I want to redesign, replace, or extend any parts. Even though Django might not be as performant as some other frameworks, such as FastAPI, it offers a well-established and stable platform for web development suitable for my Journal app.

3.4 Hierarchy Charts

Hierarchy charts are visual representations of a large problem being broken down into smaller components or sub-problems. They are charts in the form of an upside-down tree structure. Ideally, the problem should be broken down until each module is no longer than a page of code. In the charts I created, I had to separate my application into two different charts. One for my Frontend and one for my Backend because the project is quite large. In the diagram, I have broken down my problems to a reasonably small size yet not too specific such that it becomes too difficult to maintain.

3.4.1 Django Server Design:

In this hierarchy chart, I have decomposed the design of my entire Django Backend API into many smaller problems I will solve in my implementation. I will not be describing each individual problem in detail here. Still, the hierarchy chart provides a clear overview of how the different components and functionalities of the Backend system are organised. See Figure 3.1 for the chart.

3.4.2 React Frontend

In the hierarchy chart for the JavaScript Frontend, I have broken down the design of my user interface and client-side functionalities into smaller components.

These components include user authentication, data retrieval from the Backend API, the UI, and more. In this chart, it's hard to show how the functions communicate at an inter-component level. However, on top of what's designed in the hierarchy chart, data is passed through props, and the things under my state management box shown in the diagram would be accessible throughout my application by different components through the use of React Context and Context Provider. See Figure 3.2 for the chart.

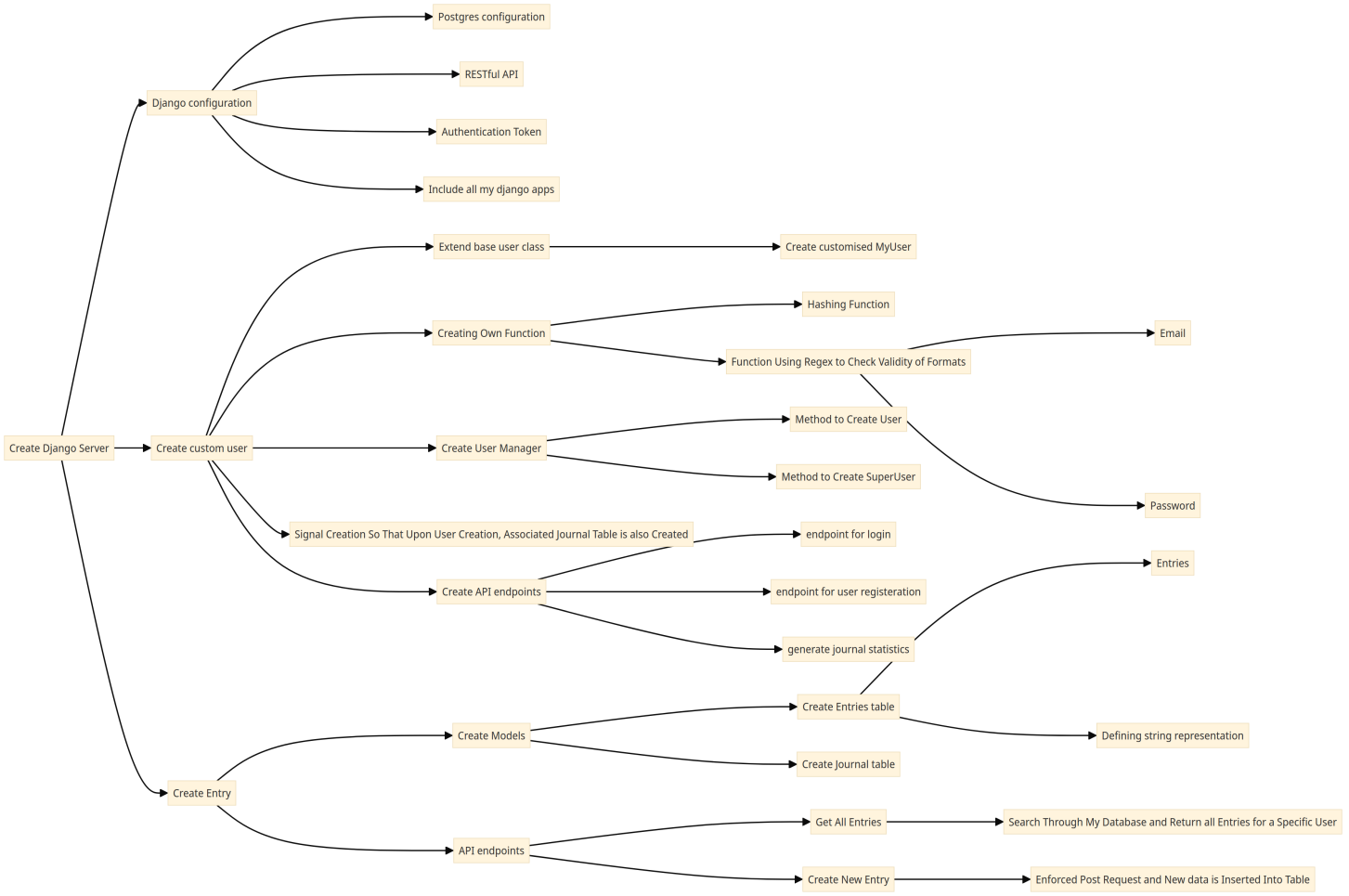


Figure 3.1: Backend Hierarchy Diagram

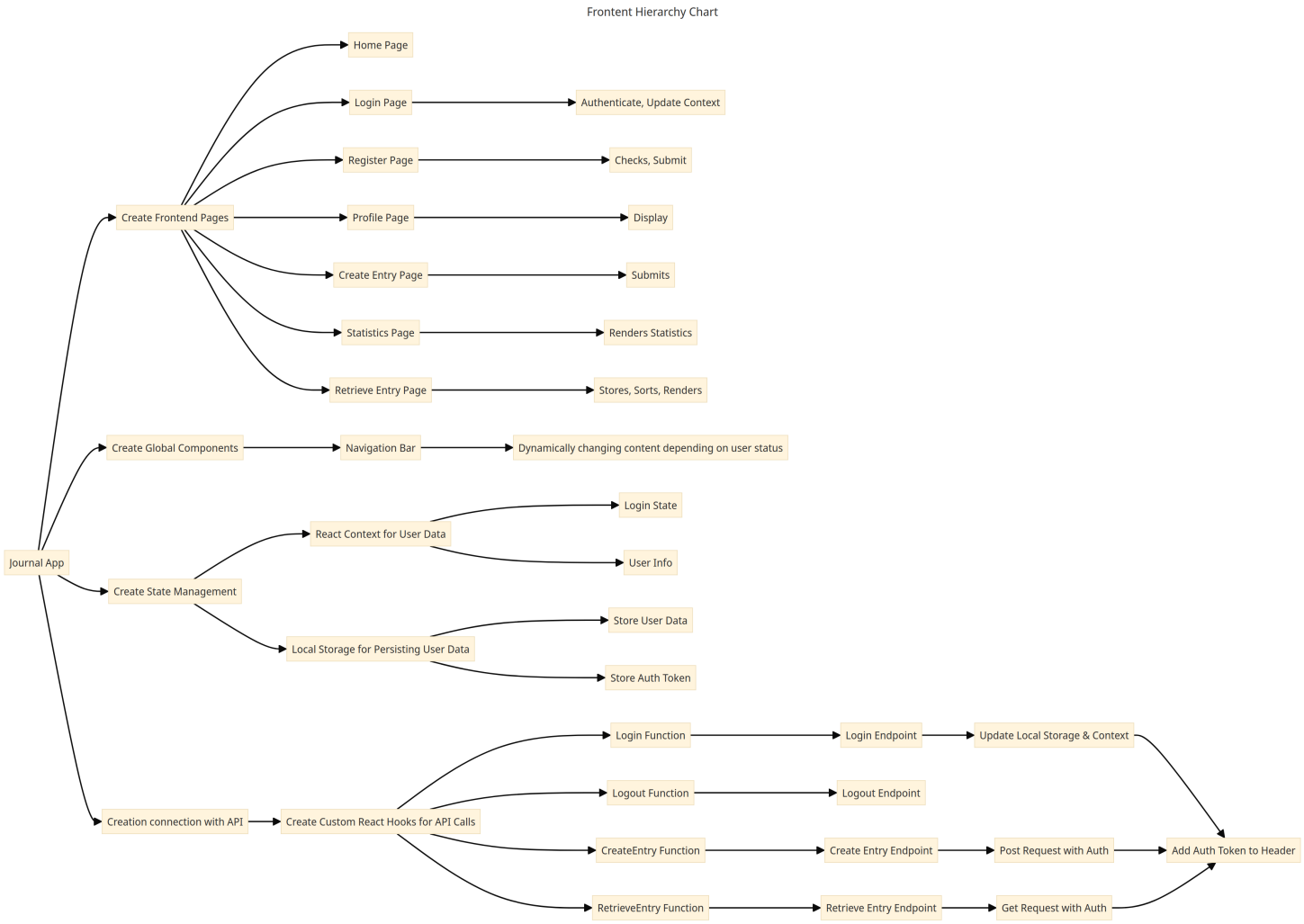


Figure 3.2: Frontend Hierarchy Diagram

3.4.3 Flowchart

Now, to bring everything together, I have designed a flowchart to depict the flow of operations across the entire system. The overall narrative of the flowchart is that the user interacts with the Frontend web pages, and it would trigger functions in the Frontend JavaScript code. These functions would perform operations such as making requests to the RESTful API endpoints exposed by the Backend or making changes to the storage in the browser. The Frontend also manipulates the storage system in the browser while the server handles the database.

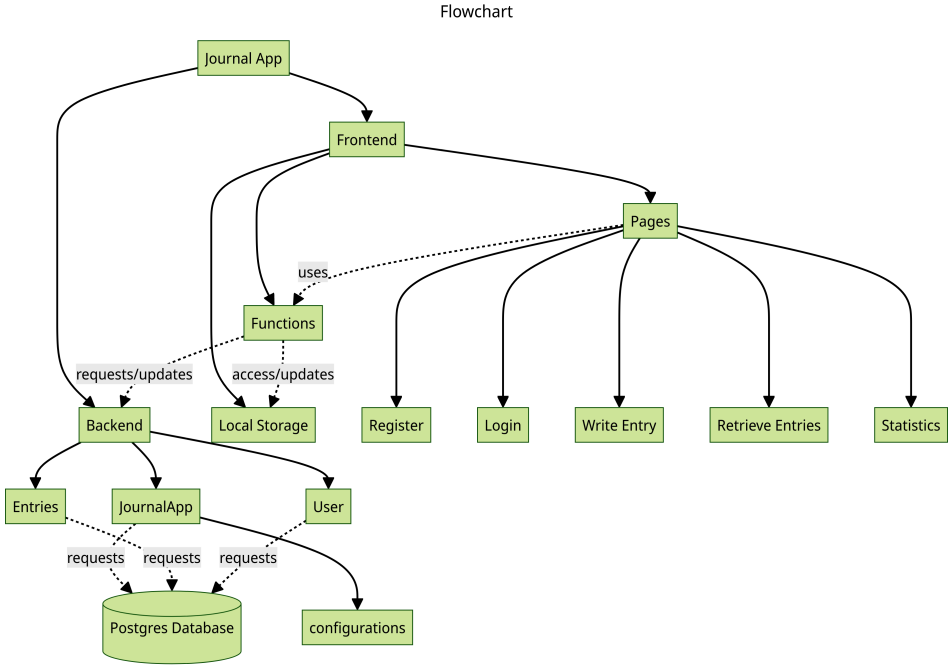


Figure 3.3: Flowchart of the System

Something to note is that users can only access webpages via HTTP(s) requests (probably through an internet browser). The Backend is not intended to be accessed by the user directly. Communication with the server happens internally through private requests. This is a good design practice as it ensures that the server is secure and reduces the risk of attacks by malicious users.

3.5 Data Structures / Data Modelling

3.5.1 Entity Relationship Modelling

Storing Data

In my application, data and the flow of data are integral to the system's overall functionality. I have decided to store the data centrally in a relational database. Central storage ensures reliability, scalability, and ease of access to data for me. This is good for my users, who expect their data to be reliably stored and easily accessed.

While it is possible to implement a distributed data storage system for a journal app, I have chosen a centralised approach for simplicity and ease of implementation. Not only is a centralised database more accessible for amateur developers like myself but there are also other reasons why centralised storage is better for my journal app. Keeping entries in a structured place means opportunities to analyse the data. If the user opts in for these features, I could perform sentiment analysis on the user's entries or generate statistics and insights based on their journaling patterns. In fact, Google provides a model for analysing sentiment, which is easily accessible through their API. This is an additional thing I may implement after my project is complete.

I have chosen PostgreSQL as the database management system for my application. Postgres is the most powerful Open-Source option available, and once configured, it provides excellent support for my development environment.

Moreover, If I were to host my application, I could easily connect it to a Postgres database through a cloud provider like Railway.

Looking through the specifications of my journal app, I have identified data that needed to be stored and created a database design in the Third Normal Form.

Entity Relationship Modelling

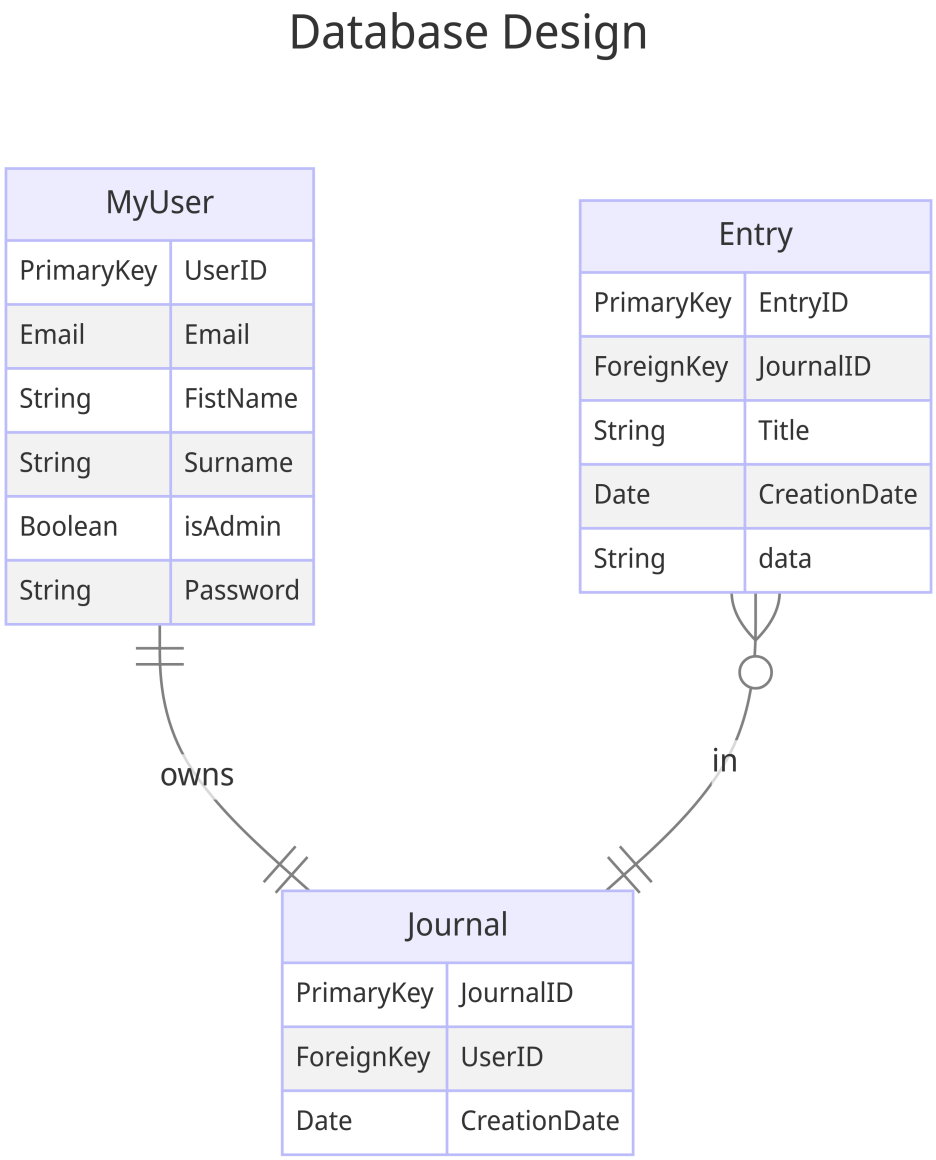


Figure 3.4: Entity Relationship Diagram

3.5.2 External Data Sources

In a bit of my code, I implemented API calls to Google Cloud natural language API to perform sentiment analysis, it is a method included in the entry class in the Backend, it can be used to generate a score for the sentiment of the user represented by a float value. This is just a gimmick feature but not core to the project.

3.5.3 OOP Model

One of the core features of Django is the built-in Object Relation Mapping, which enables me to Model my database using Objects. Initially, I explored writing SQL queries to create my database, but as my project grew in complexity, I realised that to use some powerful features provided by Django, for example, the authentication and the administrator interface, I needed to integrate Django's Object Relational Mapping functionalities with my database. With this powerful feature, the database schema can be defined using Python classes. Additionally, when it comes to interacting with defined data models, Django provides three ways to interact with the database: the Django ORM, raw SQL queries, and the admin panel. This enabled me to have the best of both worlds, the flexibility of powerful SQL queries and the ease of use of the Django ORM (with the bonus of a GUI interface).

In my models.py file, I have defined several classes that represent different objects in my project:

- MyUser: A class that represents my custom user model in Django, with additional fields and methods tailored to my project's requirements.

- MyUserManager: A class that manages MyUser class objects, including creating new user instances. MyUser has a dependency on MyUserManager for its functionality.

- Journal: A class representing a journal entry in my project, with fields such as title, content, and date. Journal has a one-to-one association with

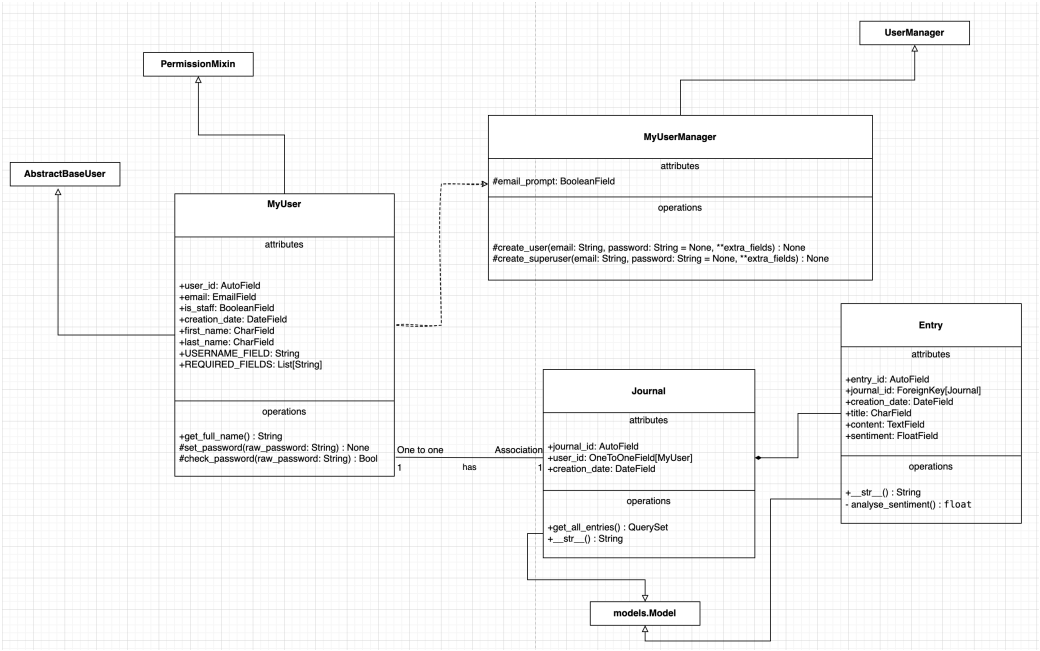


Figure 3.5: UML Class Diagram

MyUser class.

- Entry: A class with a composition relationship with the Journal class, representing a single entry within a journal.

Multiple inheritance is a feature that's not common to all OOP languages since it can lead to potential conflicts and ambiguity in the class hierarchy. In Python, it is supported, and as you can see in the UML diagram, MyUser inherits from AbstractBaseUser and PermissionsMixin. An abstract class is a class that acts as a blueprint for creating a new derived class. I have inherited Django's blueprint for creating users, and I have designed my own user class, which depends on the email field for user registration and login. The second inheritance is from the PermissionsMixin class; this enhances my user class with built-in methods for handling user permissions and authorisation, which is useful.

Both the Journal and Entry classes are inherited from the Django Model class. This allows me to utilise Django's object-relational mapping capabil-

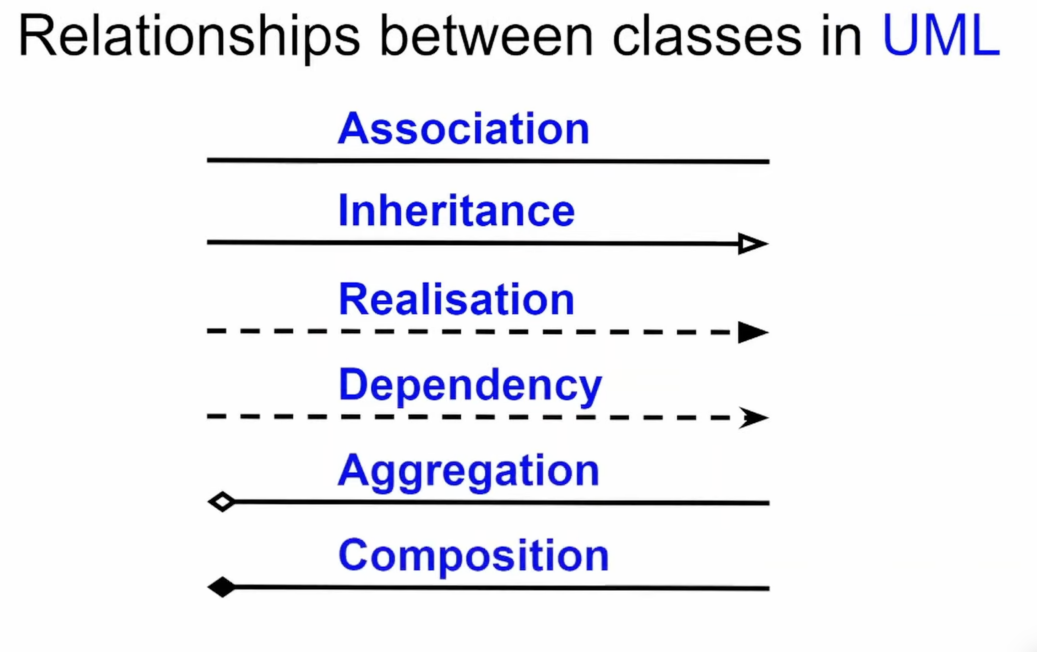


Figure 3.6: Keys explaining the arrows in the UML diagram [3]

ities to interact with the database smoothly. Entry has a composition relationship with a Journal because an entry is a part of a journal and cannot exist independently. Similarly, the Journal class has a one-to-one association with the MyUser class, indicating that each journal entry is associated with a specific user.

3.5.4 Handling JSON

JSON data are repeatedly used in my project. JSON is in the form of key-value pairs, similar to a dictionary in Python. In my project, I have used JSON data for communication and it is also just a nice way to store other information. One example off the top of my head is that I store the secret key for my application in a secret.json file which I render with Python code.

3.5.5 Typescript Interface

Conceptually Typescript interface is very similar to the Object Oriented Principal of Interface. Both are used to layout the "shape" of an object. It allows a form of abstraction and encapsulation in the code since only the blueprint of the object is defined and also many methods and properties are bundled into one. In my project, Interface is very useful to model the blueprint of the data I am expected to receive from the Backend. By defining interfaces, I can guarantee the format is as expected which increases the maintainability of the code and reduces the chance of unexpected errors.

3.6 Algorithms

In my Frontend application, once all the entries are retrieved from my database, in order to improve the experience of the user, I need there to be a way to sort and filter the entries efficiently. While it is possible to create multiple endpoints in my Backend application to return the data in multiple ways (e.g. sorted by date, filtered by category), this can result in unnecessary network requests and slower performance. To optimise the sorting and filtering of entries, I can implement an efficient algorithm in my Frontend application.

For my purpose, I will implement the sorting based on the entry submission date, although it can easily be modified to include other criteria such as the sentiment of the entries.

Choosing an effective algorithm can improve the responsiveness of the application and it can especially do so for long-term users of the application who might have a large number of entries in their journal.

Bubble sort is a simple algorithm however it has terrible time complexity making it unsuitable for large datasets. At first, I chose and implemented the QuickSort algorithm in my code since I imagined that quicksort would be efficient as it has an average time complexity of O(n log n). However, I am now rewriting my design to use Tim Sort[**?**] instead. This is because I

realised that the entries after being retrieved from my database, would be in a structured and near-sorted format. Since Tim Sort is partially insertion sort gives it the benefit of being efficient for near-sorted data, and the fact it's partially merge sort gives it the benefit of being efficient for large lists of entries with an average time complexity of O(n log n). See this graphic for a comparison of sorting algorithms.

Writing my own sorting algorithm gives me greater flexibility as I am able to define what is the key that I am deriving my my entries. In my specific case, I have chosen to implement the QuickSort algorithm for sorting the entries based on the creation date of the entry.

Here is a simple implementation I have written in Python based on the instructions from OCR A-Level Further Mathematics formula book [7], which is the first algorithm I have implemented in my project. The code is as follows:

```python
# my random array
my_array = [3, 6, 8, 10, 1, 2, 1]

def quicksort(array):
    if len(array) == 0:
        return array

    # the pivot point can either be the first element or the last element by convention
    pivot = array[0]
    leftSubarray = []
    rightSubarray = []

    for element in array[1:]:
        # iterating through the array starting from the second element to the end (because I set the pivot to be the first element, otherwise I would iterate from the first element to the second last element)
        if element < pivot:
            leftSubarray.append(element)
        else:
            rightSubarray.append(element)

    leftSubarray = quicksort(leftSubarray)
    rightSubarray = quicksort(rightSubarray)

    return leftSubarray + [pivot] + rightSubarray

# testing the quicksort function
print(quicksort(my_array))
```

Figure 3.7: QuickSort Python Implementation

3.7 User Interface

3.7.1 Wireframe Design

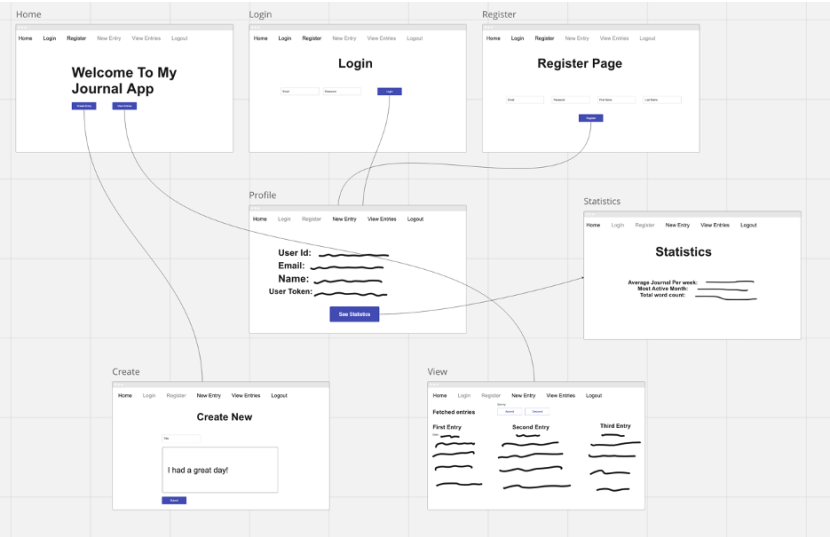


Figure 3.8: This is an overview of all the screens inside my application, it includes the home page, login page, register page, profile page, create page, and view page. There are lines connecting the pages to show the flow of the application. Using the navigation bar at the top, the user can navigate to any page from any page.

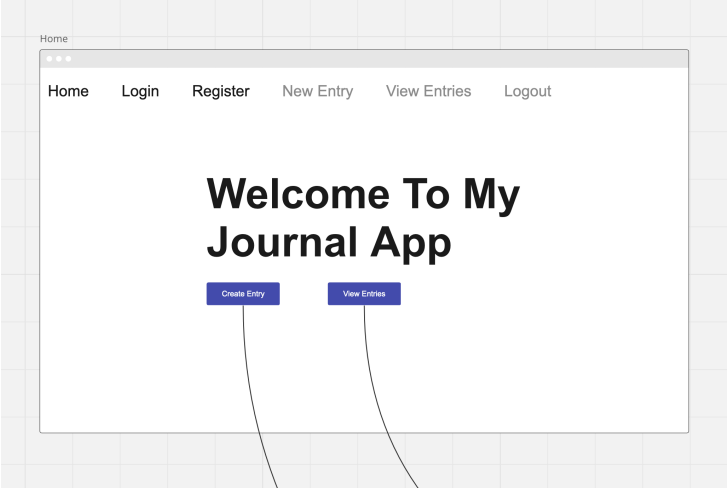


Figure 3.9: The homepage of my application, includes a navigation bar at the top, a welcome message, and buttons for creating and viewing the user's journal entries.

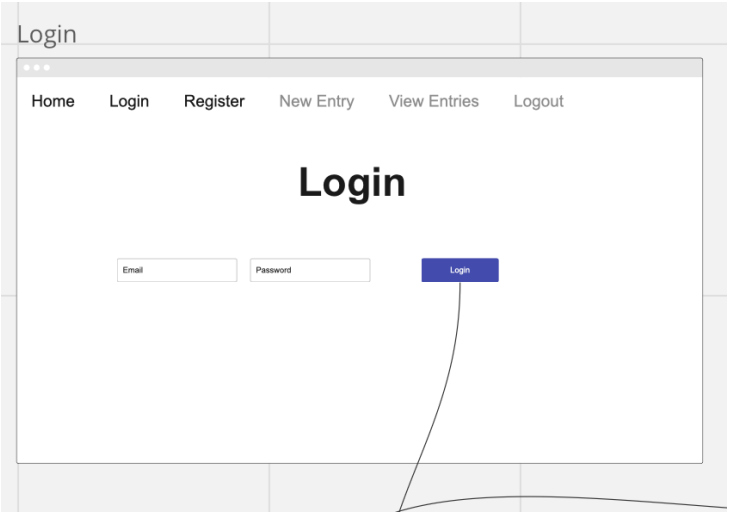


Figure 3.10: The login page of my application, includes a navigation bar at the top, a form for the user to input their email and password, and a button to submit the form. As you can see in the navigation bar, New Entry, View Entries, and Logout are all greyed out, this is assuming that the user is not logged in.

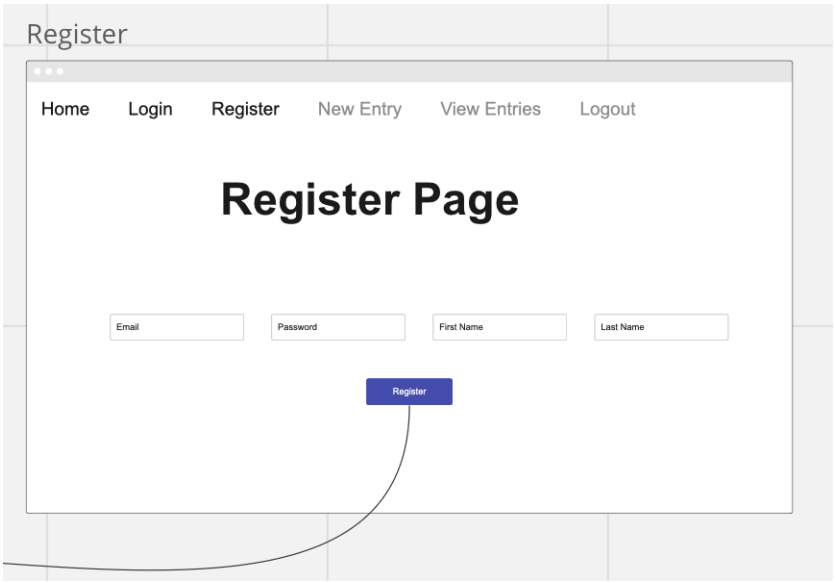


Figure 3.11: The Register page is very similar to the login page, it includes a navigation bar at the top, a form for the user inputs as well as the submission button. The registration field requires a couple more fields than the login page.

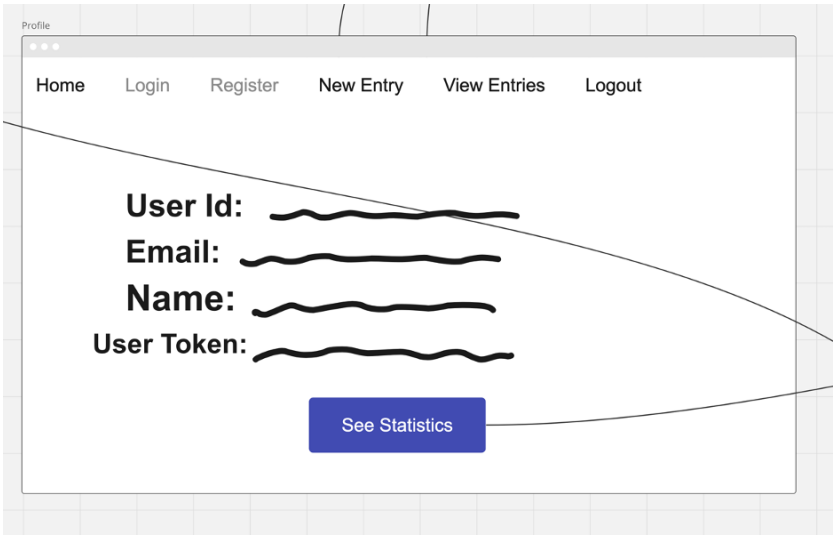


Figure 3.12: Profile page is where the user can view their profile information, as well as a place where they can opt-in for some additional features.

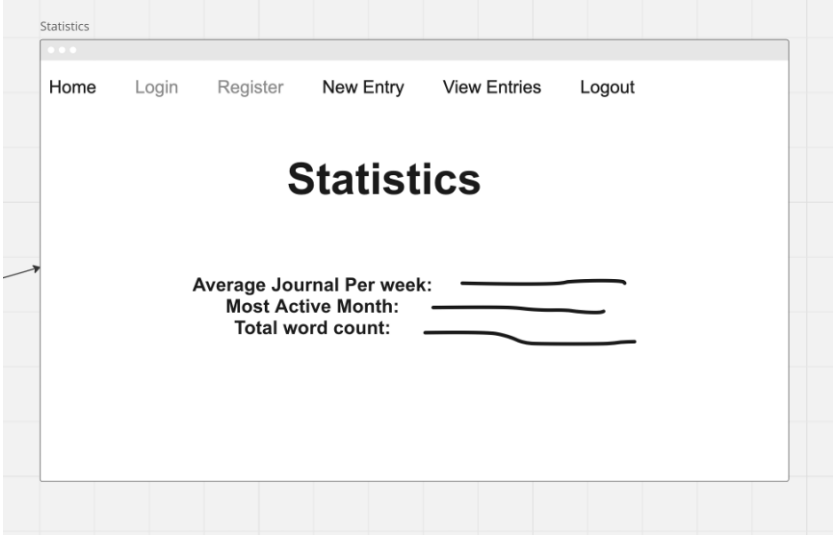


Figure 3.13: Statistics page is where the user can view statistics based on their journal entries pattern.

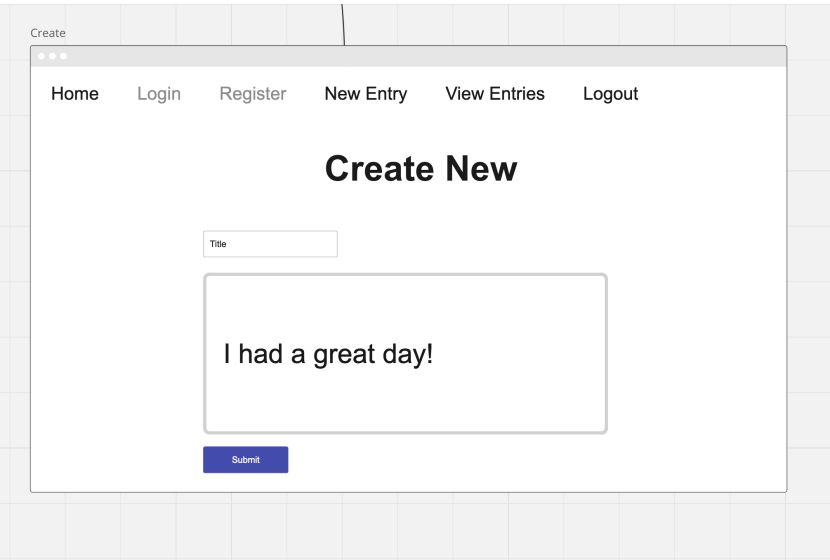


Figure 3.14: Create page is where the user can create a new journal entry. It includes a form for the user to input the title and content of the entry.

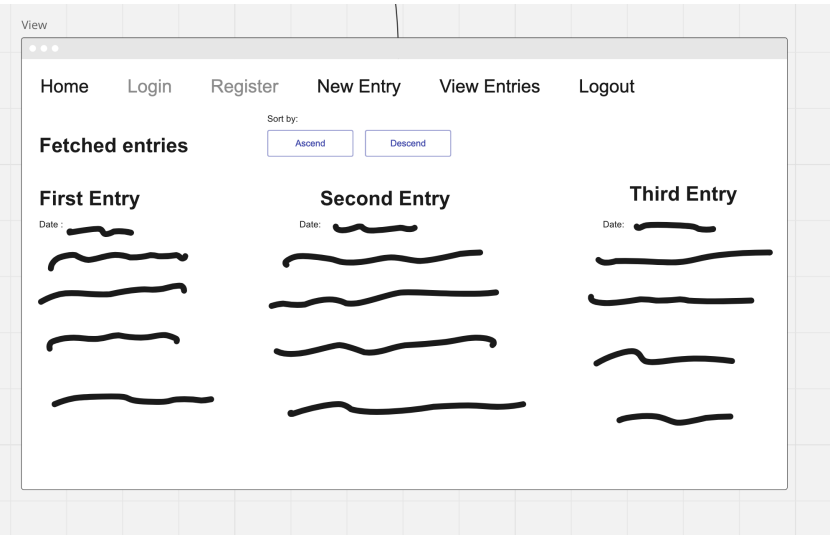


Figure 3.15: View page is where the user can view all of their journal entries. It includes buttons to enable users to sort the entries.

3.8 RESTful API Endpoints

REST effectively allows my Frontend application to directly interact with the server through HTTP requests. In this section, I will document some of the endpoints I have created in my Django Backend. One more thing to add is that I have used the Django REST framework [2] to create my API endpoints. This is a framework on top of Django specifically designed for creating RESTful APIs.

3.8.1 Serialization and deserialization of data

Serialization refers to the process of converting complex data types into a format that can easily be transmitted. For my case, I need to break down the user object into a JSON format which can then be sent to and used in my Frontend. Deserialization is the opposite process, where the JSON data

is converted back into a complex data type. I would need to process the JSON data in the Frontend, and the fact that I'm using JavaScript means that I can easily parse the JSON data, considering that JSON is a subset of JavaScript.

3.8.2 login

The login endpoint is used to authenticate the user. The user sends a POST request to the server with their email and password. The server then checks if the email and salted and hashed values of the password match with records in the database. If they do, the server generates a token and sends it back to the user. This token then can be used to authenticate future requests. For both login and register if successful serialized user data is sent back to the user along with the token.

3.8.3 register

Register is similar, but it is used to create a new user. The user sends a POST request with their email and password and other user information. The server first performs a sanity check on the format of the email via regex, then it checks if the email is already in use. If it is not, the server creates a new user with the email and password and sends back a token as well as other user details in the response.

3.8.4 get entries

This is the endpoint that retrieves all the entries of the user. The user sends a GET request, however, the caveat is that the user must be authenticated and the generated token must be attached in teh header of the request. Because the user is validated, the server knows who this user is, and can then retrieve all the entries associated with this user through the foreign key relationship in the database. The server then sends back the entries in JSON format.

3.8.5 create entry

Create entry is used to create a new entry. The user sends a POST request with the title and content of the entry. For this method, the user must also be authenticated. The server knows which user this is and through the one-to-one relationship between the user and the journal entry, the server can create a new entry and associate it with the user's journal. The server then sends back the status of the request. Otherwise, it sends back an error message.

3.8.6 get statistics

This endpoint is used to obtain statistics about a user's journal. The server generates statistics based on the user's journal entries.

3.9 Interface Logic

Here I have created flowcharts demonstrating how each of these web pages is supposed to function.

3.9.1 Login and Register Page

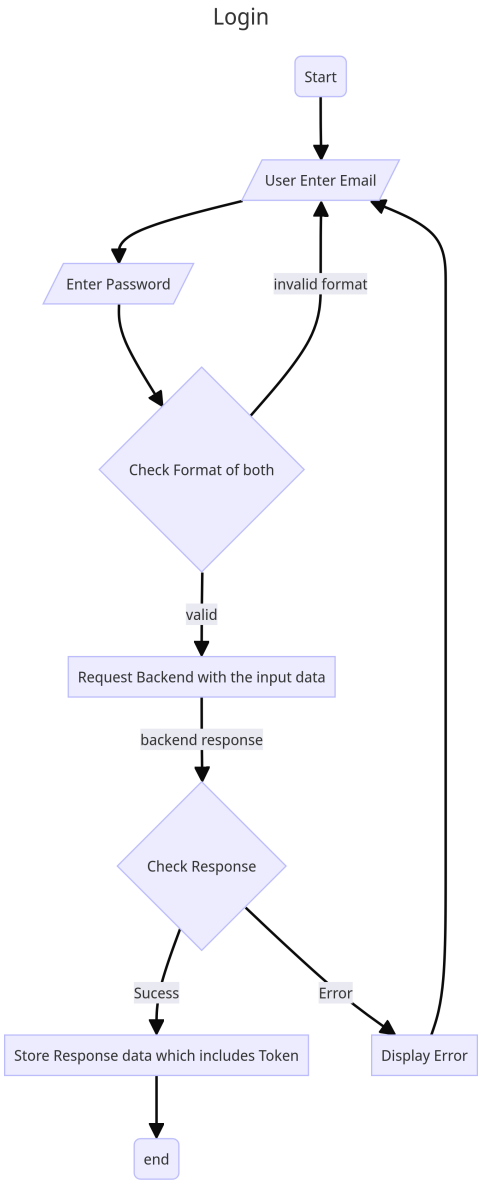


Figure 3.16: Login Page

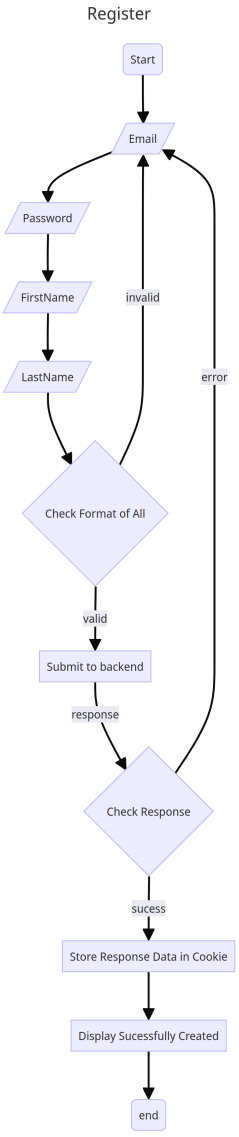


Figure 3.17: Register Page

3.9.2 Retrieve Entries Page

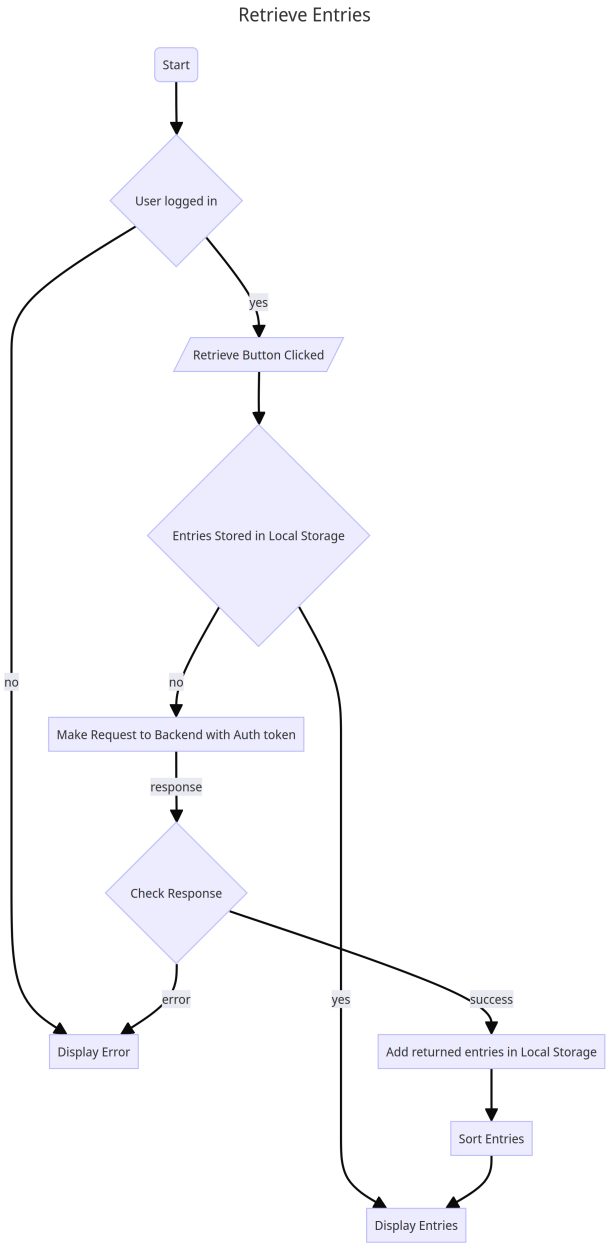


Figure 3.18: Retrieve Entries Page

3.9.3 Other

Create Entries follow a similar design to log in as it takes in user input and sends it to the server. The profile page is a page that displays the user's information, it reads the data stored in my browser's storage and then renders it. The home page is the first page the user sees when they visit the website. It includes buttons to navigate to other pages.

3.10 Generating User Statistics

The statistics page renders the statistics generated by the server. The server uses advance SQL queries to generate the statistics and then sends it back to the user. Here I will be detailing the design of those queries.

3.10.1 SQL Queries

The exact implementation can be found in the Implementation section of the report. Here I will be detailing how the queries are structured and how they work without delving much into the specifics of the code.

Most Active Month

For this feature, the logic of the query is as such:

1. Join the Journal and Entry tables on the foreign key relationship.
2. Group the entries by the month, and entry count that month using the COUNT function.
3. Order the table by the count in descending order, this way the most active month is at the top.

4. Limit the result to 1, this way only the most active month is returned.

To achieve this a couple of SQL functions are used, namely the EXTRACT, JOIN, GROUP BY, COUNT, ORDER BY, and LIMIT functions.

EXTRACT is a function available on PostgreSQL that enables me to extract only a specific detail from a date, in this case, the month. This is very handy as I could directly group the entries by the month.

As shown in the Entity Relationship Diagram 3.4, in each journal belonging to a user there are multiple entries, to collect all of these entries I could use a JOIN clause to join the two tables together, at where the foreign key of Entry is equal to the primary key of the Journal. This way I can collect all the entries associated with a user.

Since I now have the months extracted from each entry, I can now use the GROUP BY clause to aggregate the entries by the month. In SQL if I have a GROUP BY clause, I then can use the COUNT function to count the number of entries in each group. This way I can count the number of entries in each month. The defined group and the function need to both be in the SELECT clause.

Finally, I can use the ORDER BY clause to order the table by the count in descending order, this way the most active month is at the top. I can then use the LIMIT clause to limit the result to 1 to get the most active month.

Average Entry Per Week

This query involves a sub-query that gets all the weeks with the entry count and an outer query that averages this count, the logic is as such:

1. The sub-query used is very similar to the query used to calculate the most active month. What happens here is that instead of grouping all the entries by month, they are grouped by their year and ISO week

[1]. This enables me to get the number of entries for all the individual weeks the user has ever made an entry.

2. Then an outer query performs an average on the entry count column, this way I can get the average number of entries the user makes per week.

The new function used here is the AVG function which takes a field of a table as a parameter then performs an average of all the record's values in that field then an output of the average is returned. In my case, I got all the weeks and the number of entries in that week, and then I averaged the number of entries in each week to get the average number of entries per week.

Total Word count

In this query, I estimate the total word count by counting the number of spaces in the content of the entries. I basically assumed here that between every two words, there is a single space, so if I count the number of spaces +1(accounting for the first/last word), I can get the word count. The logic is as follows:

1. In a similar way I get all the entries of a journal by performing the same join.

2. Then I apply this formula to the content of the entries: "length of the content− length of the content without spaces + 1". This way I can get the number of words in the content.

3. I then sum all the words in the content of all the entries to get the total word count.

A couple of new functions used here include LENGTH, SUM, and REPLACE.

Length will be used to get the length of the content of the entries, and the length of the content without spaces.

To create the content without spaces, I can use the REPLACE function to replace all the spaces in the content with an empty string. This way I can get the length of the content without spaces.

Finally, I can use the SUM function to sum all the words in the content of all the entries to get the total word count.

3.11 Hardware & Software Requirements

Specify the hardware and software requirements for the project.

Requirement ID	Description
1	Development Machine: Apple MacBook Pro 14-inch with M1 chip, 8 GB of RAM, and 256 GB SSD. This is more than sufficient for my development, React and Python are also well supported on Apple Silicone.
2	Server for Deployment: Initially it could be configured with 2 vCPUs, 4 GB RAM, and 50 GB SSD storage which should be enough for the Backend, it should also be easily scalable.
3	End-User Devices: The app should be accessible on devices with internet connection and a web browser. This includes smartphone, laptop, even some smart IOT devices like your smart TV.

Table 3.1: Hardware Requirements for Journal App

Requirement ID	Description
1	Development Environment: Visual Studio Code with the extensions for efficient coding.
2	Terminal: I use Alacritty as my terminal emulator, the default shell on my device is zsh.
3	Package Manager: Homebrew for macOS, very useful for installing new things for example postgres was installed through this.
4	Database: PostgreSQL, as well as pgAdmin 4 for database administration and interaction.
5	Backend Framework: Python, Django REST framework, which is also compatible with my database of choice.
6	Frontend Technologies: Typescript, React, Vite.js.
7	Version Control: Git for source code management, I use it to frequency commit my code, it is also synced with Github for backup.
8	Deployment: Many options are available if I were to host my application, notably ones commonly used include Google Cloud, AWS, there are also ones built for easy deployment ones such as Railway, Heroku and Netlify.

Table 3.2: Software Requirements for Journal App

Chapter 4

Technical Implementation

In this section of my report, I will highlight sections of my code with explanations. One thing I will also discuss is best practices and patterns that I have used in my project, such as defensive programming and modularity, etc.

4.1 File Structure

See the following of the tree structure of the project. This is created using the terminal tree command on a freshly cloned version of my repository so that only important files are shown:

4.1.1 Backend

```
├── ERdiagram.png
├── Email
│   ├── __init__.py
│   ├── apps.py
│   ├── cron.py
```

```
│   ├── sendEmail.py
│   └── tests.py
├── Entries
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── test.rest
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── JournalApp
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   ├── views.py
│   └── wsgi.py
├── README.md
├── Users
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── serializers.py
│   ├── signals.py
│   ├── test.py
│   ├── test.rest
│   ├── urls.py
│   ├── utils.py
│   └── views.py
├── manage.py
├── package-lock.json
├── package.json
```

```
├── requirements.txt
└── secret.json
7 directories, 39 files
```

4.1.2 Frontend

```
├── README.md
├── index.html
├── package-lock.json
├── package.json
├── public
│   └── vite.svg
├── src
│   ├── App.css
│   ├── App.tsx
│   ├── api
│   │   ├── axios.ts
│   │   ├── getAuthTokenFromCookie.ts
│   │   └── useApi.ts
│   ├── assets
│   │   └── react.svg
│   ├── components
│   │   └── layout
│   │       ├── NavBar.css
│   │       └── NavBar.tsx
│   ├── context
│   │   └── userContext.tsx
│   ├── hooks
│   │   ├── useAuth.ts
│   │   ├── useFetchData.tsx
│   │   ├── useUser.ts
│   │   └── useUserContext.ts
│   ├── index.css
│   ├── main.tsx
│   ├── pages
│   │   ├── HomePage.tsx
```

```
│   │   ├── ProfilePage.tsx
│   │   ├── entry
│   │   │   ├── CreateEntryPage.tsx
│   │   │   ├── JournalEntriesPage.tsx
│   │   │   └── sortEntries.ts
│   │   ├── login
│   │   │   ├── LoginPage.tsx
│   │   └── register
│   │       └── RegisterPage.tsx
│   └── vite-env.d.ts
├── tsconfig.json
├── tsconfig.node.json
└── vite.config.ts
15 directories, 36 files
```

4.2 Skills

Table 4.1: Group A Model Skills Employed in Journal App Development

Model Skill	**Found in**
Complex data model in database (eg several interlinked tables)	Users/models.py and Entries/models.py from the Django project.
Hash tables, lists, stacks, queues, graphs, trees or structures of equivalent standard	Retrieved (from Backend/local storage) entries are stored as an array.
Files(s) organised for direct access	Shown above
Complex scientific/mathematical/robotics/control/business model	User session persistance as an example complex control model
Complex user-defined use of object-orientated programming (OOP) model, eg classes, inheritance, composition, polymorphism, interfaces	UML diagram in figure 3.5 is fully implemented.
Complex client-server model	Flow of data between Frontend application and Backend via RESTful API

Table 4.2: Group A Algorithms Skills Employed

Algorithm Skill	**Found in**
Cross-table parameterised SQL	Used when retrieving journal and generating statistics about the journal
Aggregate SQL functions	Generate statistics about a user's Journal
List operations	List of entries sorted and displayed in retrieveEntries page
Hashing	passwords are hashed and stored
Mergesort or similarly efficient sort	designed and implemented Quicksort but later switched to Tim Sort
Dynamic generation of objects based on complex user-defined use of OOP model	MyUser, Jounral, Entries are all generated dynamically when the user is using the app
Server-side scripting using request and response objects and server-side extensions for a complex client-server model	The core of my project
Calling parameterised Web service APIs and parsing JSON/XML to service a complex client-server model	JSON data flows between my Frontend and Backend

4.3 Key Code Segments

In this section, I will be highlighting key code segments from my project which I believe are important to understand the implementation of the project. To provide context, the source code from where these snippets are taken are also provided in the appendix.

4.4 Data Structures

4.4.1 Data Models

Data models are the backbone of my Backend server. In my design section (see figure 3.5 for the UML diagram)I have already discussed the data models that I have used in my project. Here I will provide the implementation of the data models in Django.

Modelling the User

```
1  from Django.db import models
2  from Django.contrib.auth.models import AbstractBaseUser, UserManager,
   ↪  PermissionsMixin
3
4  from .utils import clean_email, hash_password, validate_email,
   ↪  verify_password
5
6  # Create your models here.
7
8
9  def check_email_and_password(email: str, password: str | None) -> bool:
10     if password is None:
11         raise ValueError("Password is not valid")
12     if not validate_email(email):
13         raise ValueError("Email is not valid")
14     return True
15
16
17 class MyUserManager(UserManager):
18     def create_user(self, email, password=None, **extra_fields):
19         cleaned_email = clean_email(email)
20         is_valid = check_email_and_password(cleaned_email, password)
21         if is_valid:
22             user = self.model(email=cleaned_email, **extra_fields)
23             user.set_password(password)
24             user.save()
25
```

```
26      def create_superuser(self, email, password=None, **extra_fields):
27          cleaned_email = clean_email(email)
28          is_valid = check_email_and_password(cleaned_email, password)
29          if is_valid:
30              user = self.model(email=cleaned_email, **extra_fields)
31              user.set_password(password)
32              user.is_staff = True
33              user.is_superuser = True
34              user.save()
35
36
37  class MyUser(AbstractBaseUser, PermissionsMixin):
38      """A Users table used to store the data of my user a subclass
        ↪  implementing the default User model"""
39
40      user_id = models.AutoField(primary_key=True)
41      email = models.EmailField(unique=True)
42      is_staff = models.BooleanField(default=False)
43      creation_date = models.DateField(auto_now_add=True)
44      first_name = models.CharField(max_length=30, null=True)
45      last_name = models.CharField(max_length=30, null=True)
46      email_prompt = models.BooleanField(default=False, null=True)
47
48      # By convention, the manager attribute are named objects.
49      objects = MyUserManager()
50
51      USERNAME_FIELD = "email"
52      REQUIRED_FIELDS = ["first_name", "last_name"]
53
54      def get_full_name(self) -> str:
55          return f"{self.first_name} {self.last_name}"
56
57      def set_password(self, raw_password: str) -> None:
58          self.password = hash_password(raw_password)
59
60      def check_password(self, raw_password: str) -> bool:
61          return verify_password(raw_password, self.password)
62
```

Something to note here is that I have set the email field to be unique so there are no duplicate users; creation time is also auto-added.

To improve maintainability and readability, I have defined many helper

functions in a separate file called utils.py. This is an example of decomposition and in my codebase, I try to follow the Single Responsibility Principal as much as possible. This makes the code more self-contained and also makes it easier to perform unit tests on the helper functions. The code included in the appendix will contain all the relevant codes relating to what I talk about in this section of the report.

Modelling the Journal and Entries

```
from email.policy import default
from Django.db import models, connection
from Users.models import MyUser
from Django.utils import timezone

# sentiment analysis
from google.cloud import language_v2


# Create your models here.


class Journal(models.Model):
    """A Journals table pointing to the owner used to link entries
    ↪  together with the user"""

    journal_id = models.AutoField(primary_key=True)
    user_id = models.OneToOneField(MyUser, on_delete=models.CASCADE)
    creation_date = models.DateField(auto_now_add=True)

    def get_all_entries(self):
        return self.entry_set.all()

    def get_most_active_month(self):
        with connection.cursor() as cursor:
            cursor.execute(
                f"""
SELECT
  EXTRACT(MONTH FROM "Entries_entry".creation_date) AS month,
  COUNT(*) AS entry_count
FROM
```

```
  "Entries_entry"
JOIN
  "Entries_journal" ON "Entries_journal".journal_id =
  ↪  "Entries_entry".journal_id_id
WHERE
  "Entries_journal".journal_id = {self.journal_id}
GROUP BY
  month
ORDER BY
  entry_count DESC
LIMIT 1;
                """
            )
            row = cursor.fetchone()
            return row[0] if row else None

    def get_average_entries_per_week(self):
        with connection.cursor() as cursor:
            cursor.execute(
                f"""
SELECT
  AVG(entry_count)
FROM (
  SELECT
    EXTRACT(YEAR FROM "Entries_entry".creation_date) AS year,
    EXTRACT(WEEK FROM "Entries_entry".creation_date) AS week,
    COUNT(*) AS entry_count
  FROM
    "Entries_entry"
  JOIN
    "Entries_journal" ON "Entries_journal".journal_id =
    ↪  "Entries_entry".journal_id_id
  WHERE
    "Entries_journal".journal_id = {self.journal_id}
  GROUP BY
    year, week
)
LIMIT 1;
                """
            )
            row = cursor.fetchone()
            return row[0] if row else None

    def get_total_word_count(self):
        with connection.cursor() as cursor:
```

```
            cursor.execute(
                f"""
SELECT
  SUM(
    LENGTH("Entries_entry".content) -
    ↪  LENGTH(REPLACE("Entries_entry".content, ' ', '')) + 1
  ) AS total_word_count
FROM
  "Entries_entry"
JOIN
  "Entries_journal" ON "Entries_journal".journal_id =
  ↪  "Entries_entry".journal_id_id
WHERE
  "Entries_journal".journal_id = {self.journal_id};
                """
            )
            row = cursor.fetchone()
            return row[0] if row else None

    def __str__(self):
        return f"{self.user_id}'s Journal"


class Entry(models.Model):
    """An Entry table used to store the data of t journal entry and also
    ↪  tell us which journal it belongs to"""

    entry_id = models.AutoField(primary_key=True)
    journal_id = models.ForeignKey("Journal", on_delete=models.CASCADE)
    creation_date = models.DateField(default=timezone.now, editable=True)
    title = models.CharField(max_length=50, default="Untitled")
    content = models.TextField()
    sentiment = models.FloatField(default=None, blank=True, null=True)

    def __str__(self) -> str:
        return f"Entry {self.entry_id} from {self.journal_id}"

    # analyse data and return sentiment score range from -1 to 1 by
    ↪  calling google cloud api
    def analyse_sentiment(self) -> float:
        client = language_v2.LanguageServiceClient()
        document = language_v2.Document(
            content=self.content,
            ↪  type_=language_v2.Document.Type.PLAIN_TEXT
        )
```

```
114         response = client.analyze_sentiment(request={"document":
            ↪  document})
115         return response.document_sentiment.score
```

Journal and MyUser have a one-to-one Aggregate relationship, I figured it doesn't make sense for a Journal to exist, so I have specified an on_delete option to resolve this integrity error if a user is deleted. CASCADE is just an option that will delete the journal if the user is deleted. This is another defensive design. The same relationship exists between Entries and Journal as Journals are composed of Entries.

4.4.2 Generated diagram

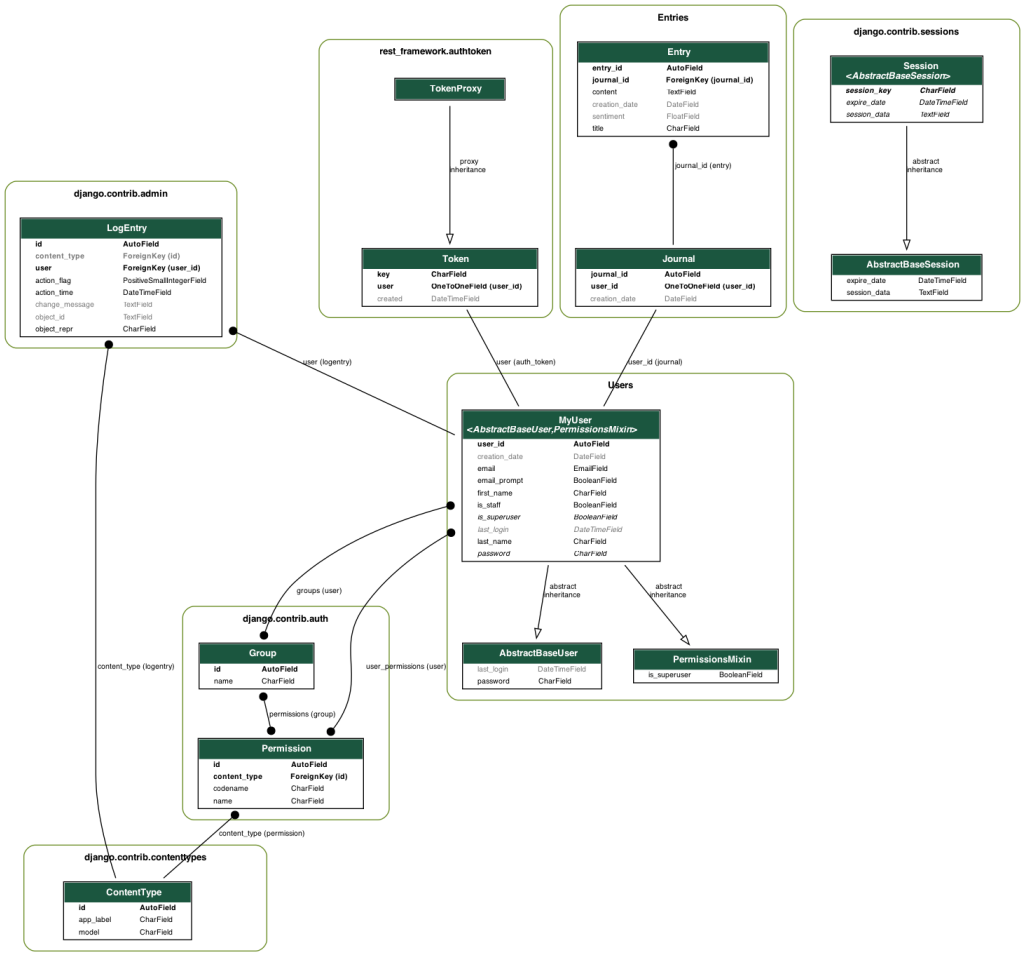


Figure 4.1: Here is a diagram generated based on my implemented Django Models using the Django-extensions library

4.4.3 Typescript Interfaces

To model JSON response provided by the Backend, I have created interfaces in Typescript. This is useful as it allows me to have type-checking in my codebase. This is especially useful when I am making API calls to the Back-

end and I want to ensure that the data I am receiving correctly matches the data I am expecting. Here are a few examples of interfaces that I have created in my Frontend more can be found in the source code:

```
  export interface Entry {
  entry_id: number;
  title: string;
  content: string;
  creation_date: string;
}
```

```
  interface registerRequestData {
  email: string;
  password: string;
  first_name: string;
  last_name: string;
}
```

```
export interface Statistics {
  most_active_month: string;
  average_entries_per_week: number;
  total_word_count: number;
}
```

4.5 Algorithms

4.5.1 Applying Tim Sort on Entries with specified Key

I have fully explained how the code works in detail in the comments of the code. How as a summary Tim Sort works by applying insertion on chunks

of the array, and then these smaller chunks are merged together. Here is the implementation:

```
import { Entry} from "./JournalEntriesPage";


export function sortEntriesByDate(entries: Entry[] | null, sortOrder:
↪  string): Entry[] {
  // defensive programming
  if (!entries) {
    return Array<Entry>();
  }
  // zipping the entries with their creation date
  const zippedEntries = entries.map((entry) => {
    return [entry, new Date(entry.creation_date).getTime()] as [Entry,
    ↪  number];
  })

  Tim Sort(zippedEntries);

  // returns the sorted entries in unzipped format
  if (sortOrder === "desc") {
    return zippedEntries.map((zippedEntry) => zippedEntry[0]).reverse();
  }
  return zippedEntries.map((zippedEntry) => zippedEntry[0]);
}


function insertionSort<Key>(
  arr: Array<[Entry, Key]>,
  startIndex: number,
  endIndex: number,
) {

  // iterate the segment of the array from startIndex to endIndex
  for (let i = startIndex + 1; i < endIndex; i++) {
    // increment the currently searching index i with each pass, and use
    ↪  j as a temprary index to compare the current element with all
    ↪  previous elements
    let j = i;

    while (j > startIndex && arr[j - 1][1] > arr[j][1]) {

      // compare the current element with every single previous element
      ↪  and keep swapping until it is in the right position
```

```
38        swap(arr, j, j - 1);
39        j--;
40      }
41    }
42  }
43
44  function swap<T>(arr: Array<[Entry, T]>, i: number, j: number) {
45    const temp = arr[i];
46    arr[i] = arr[j];
47    arr[j] = temp;
48  }
49
50
51  function merge<Key>(arr: Array<[Entry, Key]>, left: number, mid: number,
    ↪  right: number) {
52    //The two different segments sorted by insertion sort need to be
      ↪  merged, the left segment is from left to mid, and the right segment
      ↪  is from mid to right
53    const leftArr = arr.slice(left, mid);
54    const rightArr = arr.slice(mid, right);
55
56
57    // i is the index of the left segment, j is the index of the right
      ↪  segment, and k is the index of the merged segment
58    let i = 0;
59    let j = 0;
60    let k = left;
61
62    // going through all the elements of the left and right segments and
      ↪  compare them, and merge them in the right order
63    while (i < leftArr.length && j < rightArr.length) {
64
65      // basically means if the left element is smaller than the right
        ↪  element, then put the left element in the merged segment,
        ↪  otherwise put the right element in the merged segment in that
        ↪  sequence
66      if (leftArr[i][1] <= rightArr[j][1]) {
67        arr[k] = leftArr[i];
68        i++;
69      } else {
70        arr[k] = rightArr[j];
71        j++;
72      }
73      k++;
74    }
```

```
75
76    // if there are any remaining elements in the left or right segment,
      ↪  then put them in the merged segment
77    while (i < leftArr.length) {
78      arr[k] = leftArr[i];
79      i++;
80      k++;
81    }
82
83    while (j < rightArr.length) {
84      arr[k] = rightArr[j];
85      j++;
86      k++;
87    }
88  }
89
90  function Tim Sort<Key>(zippedEntries: Array<[Entry, Key]>){
91    // run means the size of the segments that will be sorted by insertion
      ↪  sort
92    const RUN = 16;
93    const n = zippedEntries.length;
94
95    // use insertion sort on all the individual runs with the size of RUN
96    for (let i = 0; i < n; i += RUN) {
97      // math.min is used so that the sorting does not go out of bounds
98      insertionSort(zippedEntries, i, Math.min(i + RUN, n));
99    }
100
101
102   // merge all the the sorted segments
103   for (let size = RUN; size < n; size = 2 * size) {
104     for (let left = 0; left < n; left += 2 * size) {
105       const mid = left + size;
106       const right = Math.min(left + 2 * size, n);
107       merge(zippedEntries, left, mid, right);
108     }
109   }
110 }
111
```

The code I have written here is designed to be reusable as I have created the sorting functions such that it sorts based on a zipped entry.

sortEntriesByDate is one of the functions that utilise the Tim Sort module, It takes in an array of entries, then creates a new array with a tuple of the entry and the creation date. This is then passed to the Tim Sort function which sorts the array based on the second index of the array (the key). The sorted array is then unzipped and returned.

In the future, the sorting modules can be used with any key by simply zipping the array with a different key and passing it to the Tim Sort function.

4.5.2 Implementation of the SQL Queries

To provide context, the following are the SQL queries that are implemented in the Django ORM as methods of the Journal model. This allows me to leverage powerful features of Python/Django enabling me to make more dynamic queries with ease. For example, I can access the journal id of the current instance of the model using self.journal_id, which can then be used directly in the query through string interpolation. In action, when a user accesses the API endpoint to get their journal statistics, self refers to the user making the request. Therefore, the query will return the statistics of the journal that the user who just made that request. Having these as methods is also a good practice as it allows for better modularity and readability.

Most Active Month (see test 7.2)

```
SELECT
  EXTRACT(MONTH FROM "Entries_entry".creation_date) AS month,
  COUNT(*) AS entry_count
FROM
  "Entries_entry"
JOIN
  "Entries_journal" ON "Entries_journal".journal_id =
  ↪  "Entries_entry".journal_id_id
WHERE
  "Entries_journal".journal_id = {self.journal_id}
GROUP BY
```

```
  month
ORDER BY
  entry_count DESC
LIMIT 1;
```

AS is used to alias the column names which makes the query more readable and efficient to write.

Average Entry Count Per Week (see test 7.1)

```
SELECT
  AVG(entry_count)
FROM (
  SELECT
    EXTRACT(YEAR FROM "Entries_entry".creation_date) AS year,
    EXTRACT(WEEK FROM "Entries_entry".creation_date) AS week,
    COUNT(*) AS entry_count
  FROM
    "Entries_entry"
  JOIN
    "Entries_journal" ON "Entries_journal".journal_id =
    ↪  "Entries_entry".journal_id_id
  WHERE
    "Entries_journal".journal_id = {self.journal_id}
  GROUP BY
    year, week
)
```

Total Word Count (see test 7.3

```
SELECT
    SUM(
```

```
      LENGTH("Entries_entry".content) -
      ↪  LENGTH(REPLACE("Entries_entry".content, ' ', '')) + 1
    ) AS total_word_count
  FROM
    "Entries_entry"
  JOIN
    "Entries_journal" ON "Entries_journal".journal_id =
    ↪  "Entries_entry".journal_id_id
  WHERE
    "Entries_journal".journal_id = {self.journal_id};
```

4.6 Modularity

Modularity is shown throughout my codebase and I try splitting code into modules when it makes sense. A couple of examples I thought were worth mentioning are the hooks I have created in my Frontend to make calls to the Backend API and also the modules I created to enable persistent login in the Frontend using context.

4.6.1 API call functions

Here I will include the code for the hooks I have created to make API calls to the Backend. This module is repeatedly reused throughout the Frontend to make calls to the Backend. For demonstration sake, I will be only including one example of my useApi hook in action.

useApi.ts

```
import { useState,  useCallback } from "react";
import  { AxiosError, AxiosRequestConfig, AxiosResponse } from "axios";
```

```
import { myApiCall } from "./axios";

// generic interface for the hook's return type
interface ApiHookState<T> {
  isLoading: boolean;
  error: AxiosError | null;
  data: T | null;
  fetchData: () => Promise<void>;
}

export default function useApi<T>(
  config: AxiosRequestConfig,
): ApiHookState<T> {
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);
  const [data, setData] = useState<T | null>(null);

  const fetchData = useCallback(async () => {
        setIsLoading(true);
        try {
          const response: AxiosResponse<T> = await myApiCall(config);
          setData(response.data);
        } catch (error: any) {
          setError(error);
        } finally {
          setIsLoading(false);
        }
      }, [config]);
      return { isLoading, error, data, fetchData };
    }
```

I have created the useApi hook because whenever I make a call to the Backend, I want to be able to retrieve information about whether the call is loading, if there is an error, and the data that is returned. These information are valuable as they can be used to conditionally render different information to the user.

This piece of code is built on top of the Axios library which is a promised[1] based library for HTTP requests. A promise exists in one of three states: pending, fulfilled, or rejected. See my implementation to see how to extract

[1]Promise is an asynchronous operation that can either succeed or fail.

additional information from my call.

To implement this feature, I have used try-catch block to handle the promise. If the promise is fulfilled, the data is stored in the data state, if the promise is rejected, the error is stored in the error state. Finally, the isLoading state is set to false to indicate that the fetch is complete. This is also an example of defensive programming.

A generic interface is used to define the return type of the hook. This is useful since it increases the reusability of the module. I can simply pass in an appropriate interface when I call the hook (like the ones I listed earlier).

There are many rules when using hooks, one of those is that I cannot define a hook inside a function. This is why I have made use of the useCallback function so that I can trigger the fetchData function whenever I need to make an API call. The fetchData function acts as my trigger and they are typically used when users press a button or when the page loads. An example is shown in the RegisterPage.tsx below.

from axios.ts

```
...
export const myApiCall: AxiosInstance = axios.create({
    baseURL: "http://127.0.0.1:8000/",
    timeout: 5000, // time before the request times out in milliseconds
    headers: {
        "Content-Type": "application/json", // I use json as the default
        ↪  content type

    },
});
```

RegisterPage.tsx

```tsx
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import useAuth from "../../hooks/useAuth";
import useApi from "../../api/useApi";
import { User } from "../../hooks/useUser";

interface registerRequestData {
  email: string;
  password: string;
  first_name: string;
  last_name: string;
}

export default function RegisterPage(): JSX.Element {
  // my states
  const [registerRequestData, setregisterRequestData] =
    useState<registerRequestData>({} as registerRequestData);
  const navigate = useNavigate();
  const { login } = useAuth();

  const { isLoading, error, data, fetchData } = useApi<User>({
    url: "/user/register/",
    method: "post",
    data: {
      email: registerRequestData.email,
      password: registerRequestData.password,
      first_name: registerRequestData.first_name,
      last_name: registerRequestData.last_name,
    },
  });

  useEffect(() => {
    if (data) {
      // redirect to profile page
      login(data);
      navigate("/profile");
    }
  }, [data]);
  const onRegisterFunc = () => {
    console.log("Registering...");
    fetchData(); // triggers myApiCall
```

```
42    };
43    return (
44      <div>
45        <input
46          placeholder="email"
47          value={registerRequestData.email}
48          onChange={(e) =>
49            setregisterRequestData({
50              ...registerRequestData,
51              email: e.target.value,
52            })
53          }
54        />
55        <input
56          type="password"
57          placeholder="password"
58          value={registerRequestData.password}
59          onChange={(e) =>
60            setregisterRequestData({
61              ...registerRequestData,
62              password: e.target.value,
63            })
64          }
65        />
66        <input
67          placeholder="first name"
68          value={registerRequestData.first_name}
69          onChange={(e) =>
70            setregisterRequestData({
71              ...registerRequestData,
72              first_name: e.target.value,
73            })
74          }
75        />
76        <input
77          placeholder="last name"
78          value={registerRequestData.last_name}
79          onChange={(e) =>
80            setregisterRequestData({
81              ...registerRequestData,
82              last_name: e.target.value,
83            })
84          }
85        />
86        {isLoading && <div>Loading...</div>}
```

```
87        {error && <div>{error.message}</div>}
88        {error && <div>{JSON.stringify(error.response?.data)}</div>}
89        {data && (
90          <div>
91            Registered successfully
92            <div>Welcome {data.first_name}</div>
93          </div>
94        )}
95
96        <br />
97
98        <input
99          disabled={isLoading}
100         type="button"
101         value="Register"
102         onClick={onRegisterFunc}
103       />
104     </div>
105   );
106 }
```

Before the button is clicked I have made use of the useState hook to store the user's input, and as the user types in the input field, the state is updated.

In this example by using my useApi hook, as the button is clicked (see line 102), onRegisterFunc is called which triggers the fetchData function, which activates my logic of the useApi call, submitting all the information stored in my states to my Backend. My useApi function provides live feedback about the state of the fetch as it happens, and that information can help me dynamically display information to the user.

In line 99, I have set disabled to isLoading so that the button is disabled when the fetch is loading, which is a defensive programming technique to prevent multiple clicks on the button.

In addition, see line 86 for a couple of examples of how I conditionally render different information based on the state of the fetch. Conditional rendering is another technique I employed throughout my Frontend to provide a better user experience as well as to defensively program my application.

4.6.2 Persistent Login Session

To enable persistent login sessions in my React app, the browser's local storage as well as context is used. Context is used so that the user's information can be stored and accessed throughout the application, avoiding the need for prop drilling.

useUser.ts

```
import Cookies from "universal-cookie";
import useUserContext from "./useUserContext";

export interface User {
    user_id: number;
    email: string;
    first_name: string;
    last_name: string;
    authToken?: string;
    email_prompt: boolean;
  }
export default function useUser() {
    const { updateUser } = useUserContext();

    const cookies = new Cookies();

    const addUser = (user: User) => {
        cookies.set("user", user, { path: "/",
        secure: true,
        sameSite: "strict"

    });
        updateUser(user);
        console.log("User added");
    }

    const removeUser = () => {
```

[1]In react information is passed through components from parent to child, Prop Drilling is when a certain piece of information is passed through many components. This is bad practice as it clogs up the code making it difficult to maintain.

```
28          cookies.remove("user");
29          updateUser(null);
30          console.log("User removed");
31      }
32      const getUser = (): User | undefined  => {
33          return cookies.get("user");
34      }
35
36      return {
37          addUser,
38          removeUser,
39          getUser
40      }
41  }
```

useUser is a hook I have defined that governs the storage and retrieval of user information. It is used to store the user's information in the user's browser in the form of cookies, and also to retrieve the user's information storage. This is useful as it allows me to persist even if the user closes the tab or refreshes. Cookie configuration is also set to be secure and sameSite to strict to prevent CSRF attacks.

See lines 29 and 23, the context is also updated at the same time the cookie is updated. This is to ensure that the user's information is accessible throughout the application.

userContext.tsx

```
1  import Cookies from "universal-cookie";
2  import { User } from "../hooks/useUser";
3  import React, { createContext, useState, ReactNode, useEffect } from
   ↪  "react";
4
5  export interface UserContextType {
```

[1]I have chosen cookies instead of local storage for security reasons, see defensive programming section.

```
  user: User | null;
  updateUser: (user: User | null) => void;
}

export const UserContext = createContext<UserContextType | undefined>(
  undefined
);

export const UserProvider: React.FC<{ children: ReactNode }> = ({
  children,
}) => {
  const cookies = new Cookies();
  const [user, setUser] = useState<User | null>(() => {
    // prevent my user from being lost on refresh
    const savedUser = cookies.get("user");
    if (savedUser) {
      return savedUser;
    }
  });

  const updateUser = (newUser: User | null) => {
    setUser(newUser);
    console.log("update user for context: ", newUser, user);
  };

  useEffect(() => {
    if (user) console.log("user context: ", user);
  }, [user]);

  return (
    <UserContext.Provider value={{ user, updateUser }}>
      {children}
    </UserContext.Provider>
  );
};
```

useUserContext.ts

```
import { useContext} from "react";
import { UserContext, UserContextType  } from "../context/userContext";

export default function useUserContext() {
    const context = useContext(UserContext) as UserContextType;

    if (context === undefined) {
        throw new Error("useUser must be used within a UserProvider");
    }

    return context;

}
```

As you can see here in my provider I have user and updateUser as the context value, these are the two functions that are used to retrieve and update the user's information. I have created a useUserContext hook to access the context to make the code clearer and more modular.

main.tsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.tsx";
import "./index.css";
import { UserProvider } from "./context/userContext.tsx";
// import { QueryClient, QueryClientProvider } from "react-query";
import { BrowserRouter } from "react-router-dom";

ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>
    <UserProvider>
      <BrowserRouter>
        <App />
      </BrowserRouter>
    </UserProvider>
```

```
  </React.StrictMode>
);
```

This is the entry point of my application. I have wrapped my entire app components and routes in the UserProvider so that the user's information can be accessed throughout the entire application.

useAuth.ts

```
import { User } from "./useUser";
import useUser from "./useUser";

export default function useAuth() {
    const { getUser, addUser, removeUser } = useUser();

    const login = (user: User) => {
        addUser(user);
    }

    const logout = () => {
        removeUser();
    }

    const isAuthenticated = () => {
        return getUser() !== undefined;
    }

    return {
        login,
        logout,
        isAuthenticated
    }}
```

useAuth is used to login and logout of the user. These are created to make the code more modular as it uses the useUser to perform certain actions on the user's information. This is also an example of the Single Responsibility

Principal as the useAuth hook is only responsible for authentication and nothing else.

4.6.3 Virtual Environment

Virtual environment is used to manage the dependencies of the project. This is useful as it allows me to keep the dependencies of the project separate from the dependencies of the system. I have also included the requirements for both the Django and React app in the root of the projects. This is also useful for deployment as it allows me to easily deploy the project on a server without having to worry about the dependencies.

4.7 Defensive Programming / Robustness

In this section, I will be discussing the defensive programming techniques and preventions I have implemented in my project to ensure that the application is robust and secure.

4.7.1 Sanity checking

```
import re
def validate_email(email: str) -> bool:
  return re.match(r"[^@]+@[^@]+\.[^@]+", email) is not None


def clean_email(email: str) -> str:
cleaned_email = re.search(r"[^@]+@[^@]+\.[^@]+", email)
if cleaned_email is not None:
    parsed_email =  cleaned_email.group(0)
    return parsed_email.lower().strip()
else:
    return ""
```

These functions are used in the User model to validate and clean the email before it is stored in the database. The regex ensures the email format is in the form of email in the correct form. This simple regex basically ensures that the email is in the form of a@b.c where a, b, c are characters that are not @. This is a simple form of input validation to ensure that the email is in the correct format before it is stored in the database.

For cleaning the data I extract the text matching that regex then convert it into lowercase and strip any leading or trailing whitespace. This is to ensure that the email is stored in a consistent format in the database.

4.7.2 Password Hashing

```
import hashlib
import os


def hash_password(password: str) -> str:
    salt = os.urandom(16)
    salted_password = salt + password.encode("utf-8")

    sha256_hash = hashlib.sha256()
    sha256_hash.update(salted_password)
    hashed_password = sha256_hash.hexdigest()

    return f"{salt.hex()}:{hashed_password}"


def verify_password(password: str, hashed_password: str) -> bool:
    salt, stored_hash = hashed_password.split(":")

    salted_password = bytes.fromhex(salt) + password.encode("utf-8")
    sha256_hash = hashlib.sha256()
    sha256_hash.update(salted_password)
    provided_hash = sha256_hash.hexdigest()

    return provided_hash == stored_hash

```

A salt is basically a random string that is added to the password before it's hashed and it prevents the same password from having the same hash value, hence reducing the risk of a dictionary attack or rainbow table attack. In my database I store the salt and the hashed password in the format salt(in hex):hashed_password. This is to ensure that the salt is stored along with the password so that it can be used to verify the password when the user logs in.

Verify password is used to verify the password when the user logs in. It takes the input password, hashes it with the stored salt, and sees if the hashes match. If they match then the password is correct, if not then the password is incorrect.

4.7.3 SQL Injection

SQL injection is protected in my codebase as I have used Django's ORM to make queries to the database. Django's ORM automatically escapes the input to prevent SQL injection. This prevents malicious users from injecting SQL code into the database.

4.7.4 Cross Site Request Forgery (CSRF)

To prevent CSRF attacks, Django has a built-in middleware that checks the CSRF token in the request. This is enabled by default in Django and is used to prevent CSRF attacks. This is a good practice as it prevents malicious users from making requests on behalf of the user.

4.7.5 Cookies vs Local Storage

Sensitive information like user data and authentication tokens are stored in cookies instead of local storage. This is because cookies are more secure

than local storage, because Local storage is vulnerable to XSS attacks, while cookies are not.

For retrieved journal entries, however, I have used local storage as it is not sensitive information. Each website has access to 5MB of data that can be stored in local storage, so caching the journal entries in local storage is a good practice as it reduces the number of requests made to the server.

4.7.6 Rate Limiting

```
  # from JournalApp/settings.py
  REST_FRAMEWORK = {
    "DEFAULT_AUTHENTICATION_CLASSES": [
        "rest_framework.authentication.BasicAuthentication",
        "rest_framework.authentication.SessionAuthentication",
    ],
    "DEFAULT_THROTTLE_CLASSES": [
        "rest_framework.throttling.AnonRateThrottle",
        "rest_framework.throttling.UserRateThrottle",
    ],
    "DEFAULT_THROTTLE_RATES": {
        "anon": "8/minute",
        "user": "30/minute",
    },
}
```

With Django Rest Framework, I can configure the rate limit of the API to prevent DoS attacks. In this example, I have set the rate limit to 8 requests per minute for anonymous users and 30 requests per minute for authenticated users.

4.7.7 secret key management

I have stored my sensitive details like logins or keys in a secret.json file, which is included in my git ignore file. This is to prevent the secret key from being

exposed in the repository.

4.8 Error Handling

Error handling is used throughout my application, here's a typical example from Users/views.py:

```
@api_view(["POST"])
def login_view(request) -> Response:
    try:
        user = MyUser.objects.get(email=request.data["email"])
        if user.check_password(request.data["password"]):
            token, created = Token.objects.get_or_create(user=user)
            user_data = MyUserSerializer(user).data
            user_data["token"] = token.key
            response_data = {
                "user_id": user.user_id,
                "email": user.email,
                "first_name": user.first_name,
                "last_name": user.last_name,
                "authToken": token.key,
            }
            return Response(response_data, status=status.HTTP_200_OK)
        else:
            return Response(
                {"error": "Invalid password"},
                ↪  status=status.HTTP_400_BAD_REQUEST
            )

    except MyUser.DoesNotExist:
        return Response(
            {"error": "User does not exist"},
            ↪  status=status.HTTP_404_NOT_FOUND
        )
```

Chapter 5

Testing

5.1 Test Strategy

Video demonstrations of the app will be provided to show the app in action, this will provide a complete overview of the app and its features. This is my acceptance testing as it will show the app working and meeting the original requirements.

The primary testing strategy for my Backend project would be Unit testing to see if all the smaller components of my app work as intended. Django provides its testing framework built on top of Python's unittest module. I will be using this to test my models and the methods within them, as well as the utility functions I have created in my app. Normal, boundary, and edge cases will be tested where applicable. In addition, the PostgreSQL database will also be shown in a video.

For the API endpoints, I will be using REST client provided by Visual Studio Code to make HTTP requests to the server and compare the result with my anticipated result. I will be doing this rather than using Django's own testing framework because I want to model real-world usage of my app by making legitimate HTTP requests to modify the database.

For my Frontend Interface, I will be testing the app manually by interacting with the components to see if the app behaves as expected. This also acts as my integration testing as I will be testing the app as a whole - the Frontend depends on the Backend.

5.1.1 Unit Testing

User

Here are the unit tests I have written for the User model in my app, the purpose of the tests are evident from the name of the tests.:

```
from django.test import TestCase
from django.core.exceptions import ValidationError
from Users.models import MyUser
from .utils import validate_email, clean_email, hash_password,
↪  verify_password

from django.test import TestCase
from .utils import hash_password, verify_password, validate_email,
↪  clean_email


class UtilityFunctionTests(TestCase):

    def test_hash_password(self):
        password = "testpassword123"
        hashed_password = hash_password(password)
        self.assertIsInstance(hashed_password, str)
        self.assertTrue(":" in hashed_password)

    def test_verify_password(self):
        password = "testpassword123"
        hashed_password = hash_password(password)
        self.assertTrue(verify_password(password, hashed_password))

    def test_verify_password_incorrect(self):
        password = "testpassword123"
        incorrect_password = "wrongpassword"
        hashed_password = hash_password(password)
```

```
27          self.assertFalse(verify_password(incorrect_password,
            ↪  hashed_password))
28
29      def test_validate_email_valid(self):
30          valid_email = "test@test.com"
31          self.assertTrue(validate_email(valid_email))
32
33      def test_validate_email_invalid(self):
34          invalid_email = "not-an-email"
35          self.assertFalse(validate_email(invalid_email))
36
37      def test_clean_email_valid(self):
38          valid_email = "test@test.com"
39          self.assertEqual(clean_email(valid_email), valid_email)
40
41      def test_clean_email_all_caps(self):
42          test_email = "TEST@TEST.COM"
43          self.assertEqual(clean_email(test_email), test_email.lower())
44
45      def test_clean_email_with_extra_characters(self):
46          valid_email = "      test@test.com  "
47          self.assertEqual(clean_email(valid_email), "test@test.com")
48
49
50  class MyUserModelTests(TestCase):
51
52      def test_create_user(self):
53          email = "test@test.com"
54          password = "password"
55          user: MyUser = MyUser.objects.create_user(email=email,
            ↪  password=password)
56          self.assertEqual(user.email, email)
57          self.assertTrue(user.check_password(password))
58
59      def test_create_user_invalid_email(self):
60          email = "not-an-email"
61          password = "password"
62          with self.assertRaises(ValueError):
63              MyUser.objects.create_user(email=email, password=password)
```

```
❯ python manage.py test Users
Found 10 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..........
----------------------------------------------------------------------
Ran 10 tests in 0.019s

OK
Destroying test database for alias 'default'...
```

Figure 5.1: Unit Test Result for User App in Django

Journal

Here are the unit tests I have written for the Journal model in my app:

```python
from django.test import TestCase
from .models import Journal, Entry
from Users.models import MyUser


class JournalModelTests(TestCase):
    def setUp(self):
        self.user = MyUser.objects.create_user(
            email="test@test.com", password="password"
        )

    # test if an entry asssociated with the user is created automatically using a signal
    def test_create_journal(self):
        journal = Journal.objects.get(user_id=self.user)
        self.assertEqual(journal.user_id, self.user)
        self.assertEqual(journal.get_all_entries().count(), 0)

    def test_create_entry(self):
        journal = Journal.objects.get(user_id=self.user)
        entry = Entry.objects.create(journal_id=journal, content="first entry")
        self.assertEqual(entry.journal_id, journal)
        self.assertEqual(journal.get_all_entries().count(), 1)

    def test_get_all_entries(self):
        journal = Journal.objects.get(user_id=self.user)
```

```
        entry1 = Entry.objects.create(journal_id=journal, content="first
        ↪  entry")
        entry2 = Entry.objects.create(journal_id=journal, content="second
        ↪  entry")
        self.assertEqual(journal.get_all_entries().count(), 2)
```

```
❯ python manage.py test Entries
Found 3 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
...
----------------------------------------------------------------------
Ran 3 tests in 0.018s

OK
Destroying test database for alias 'default'...
```

Figure 5.2: Unit Test Result for Entries App in Django

5.2 Endpoints

To model real-world usage, I will be making HTTP requests to the server and compare the result with my anticipated result. Here is a list of views created in my app:

```
# from entries/urls.py
urlpatterns = [
    re_path("sample/", views.sampleEntry, name="sample entry"),
    re_path("testUser/", views.test_user, name="test user"),
    re_path("getEntries/", views.get_all_entries, name="get all
    ↪  entries"),
    re_path("createEntry/", views.create_entry, name="create entry"),
    re_path(
        "getScheduledUsers/",
        views.get_daily_scheduled_email_users,
        name="get scheduled email users",
    ),
    re_path("deleteEntry/", views.delete_entry, name="delete entry"),
    re_path(
```

```
        "getJournalStatistics/",
        views.get_journal_statistics,
        name="get entry statistics",
    ),
]
```

```
# from users/urls.py
urlpatterns = [
    re_path("register/", views.register_view, name="api_register"),
    re_path("login/", views.login_view, name="api_login"),
    re_path("logout/", views.logout_view, name="api_logout"),
    re_path("test_user/", views.test_user, name="test_user"),
    # re_path("update_email_prompt/", views.update_email_prompt,
    ↪  name="email_prompt"),
    # path("session/", views.session_view, name="api_session"),
]
```

5.2.1 Endpoint tests with Rest Client

```
❯ python manage.py runserver

Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 26, 2024 - 17:25:23
Django version 5.0.1, using settings 'JournalApp.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Once the server is running, I began to make requests, see Appendix for the results

5.2.2 Test Cases Summary

Test Case	**Outcome**	**Figure Reference**
Register a new user that already exists	Successful	Fig. 7.4, 7.5
Register a new user that does not exist	Successful	Fig. 7.6, 7.7
Register a new user with an invalid email	Successful	Fig. 7.8, 7.9
Login with a valid user	Successful	Fig. 7.10, 7.11
Login with an invalid user	Successful	Fig. 7.12, 7.13
Login with an invalid password	Successful	Fig. 7.14, 7.15
Login with an invalid email	Successful	Fig. 7.16, 7.17
Test user authentication with a valid token	Successful	Fig. 7.18, 7.19
Test user authentication with an invalid token	Successful	Fig. 7.20, 7.21
Get samples of entries	Successful	Fig. 7.22, 7.23
Test user in entry app	Successful	Fig. 7.24, 7.25
Create a new entry	Successful	Fig. 7.26, 7.27
Get entries	Successful	Fig. 7.28, 7.29
Delete entries with an invalid id	Successful	Fig. 7.30, 7.31
Delete an entyr with a valid id	Successful	Fig. 7.32, 7.33
Get journal statistics	Successful	Fig. 7.34, 7.35

Table 5.1: Summary of API Test Cases, Outcomes, and Figure References

5.3 Frontend Interface

5.3.1 Home Page

Logged Out View

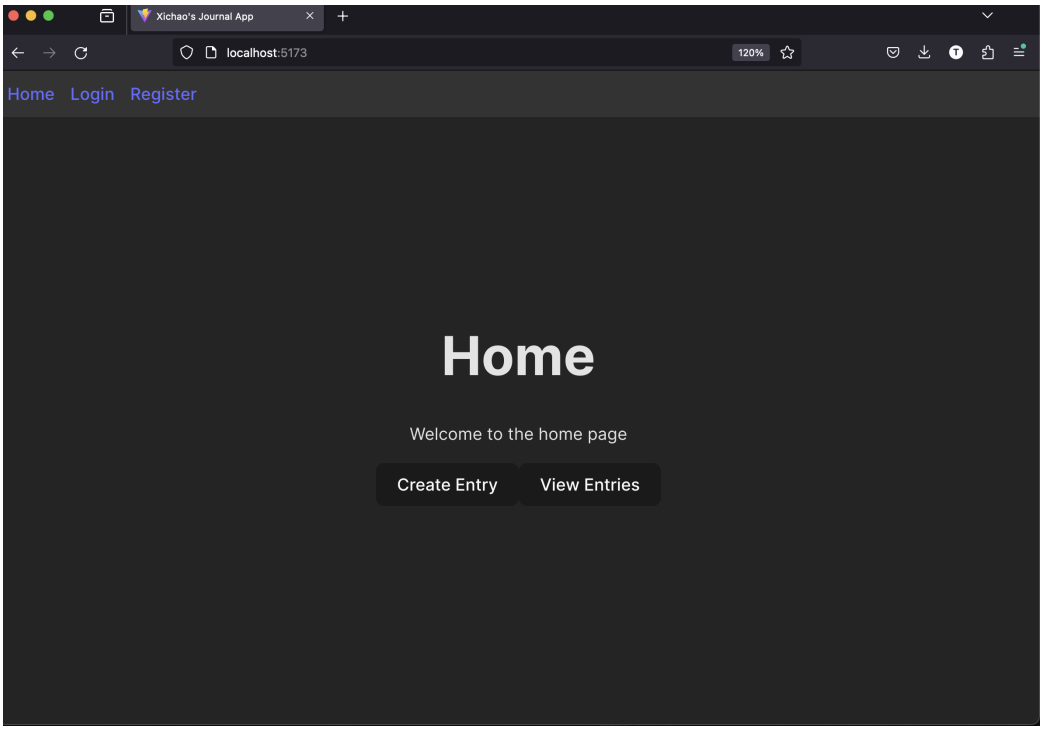


Figure 5.3: Home page view when logged out. The navigation bar shows options available to non-authenticated users.

Logged In View

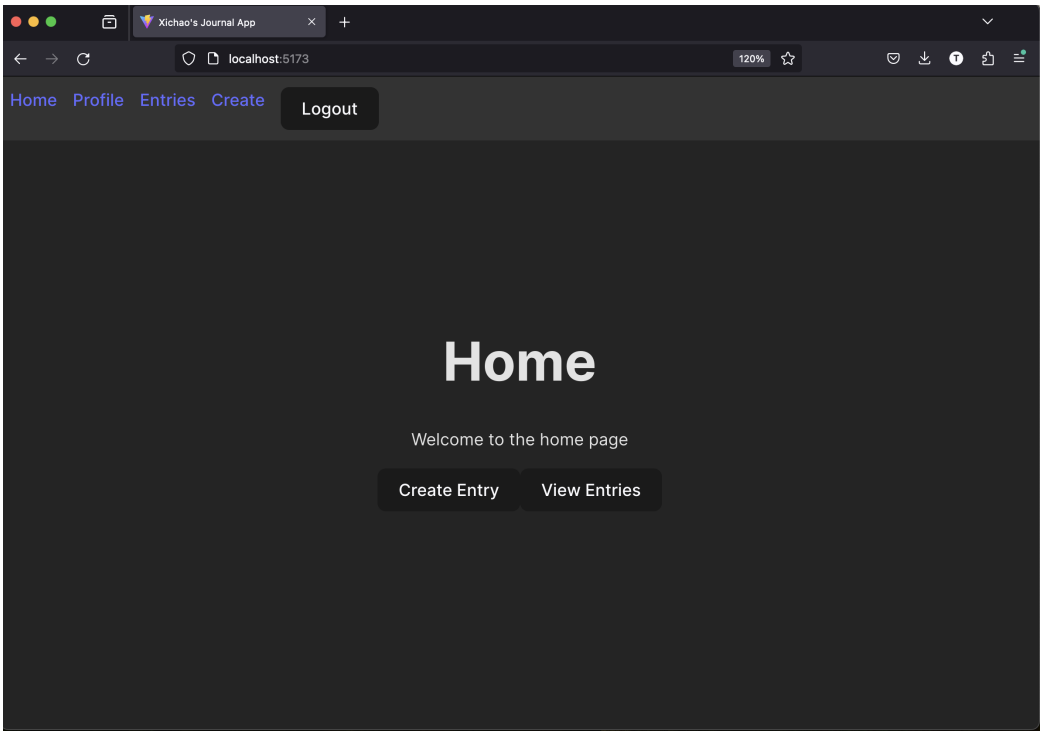


Figure 5.4: Home page view when logged in. The navigation bar changes to show user-specific options.

5.3.2 Authentication Pages

Login Page

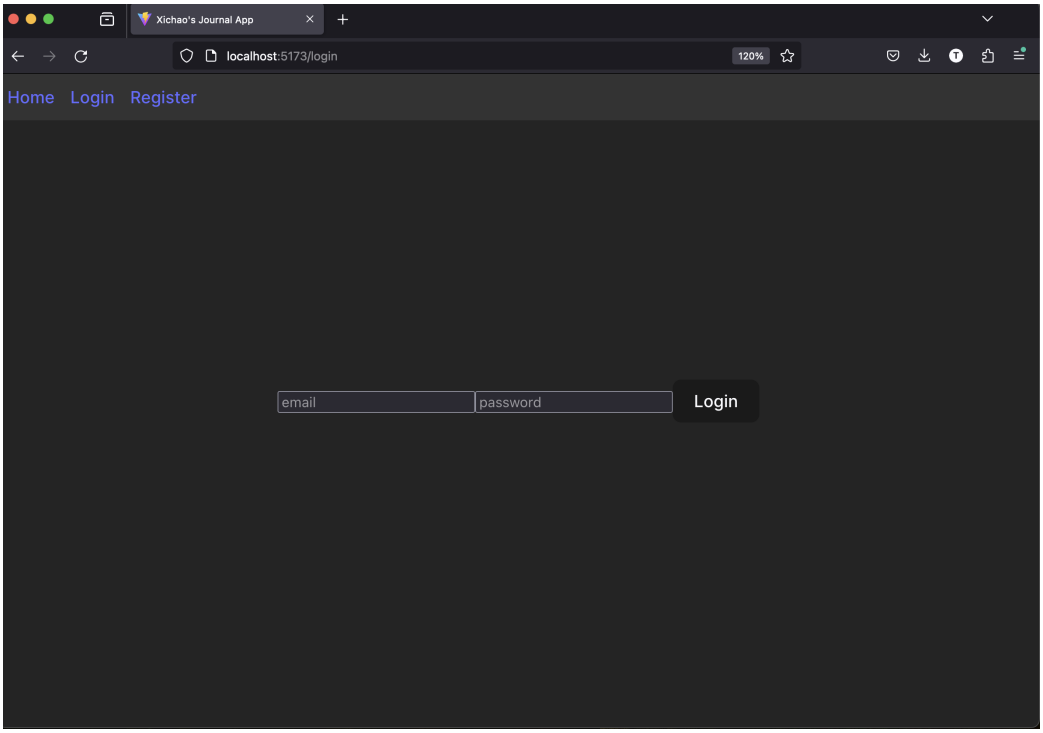


Figure 5.5: Login page view.

Registration Page

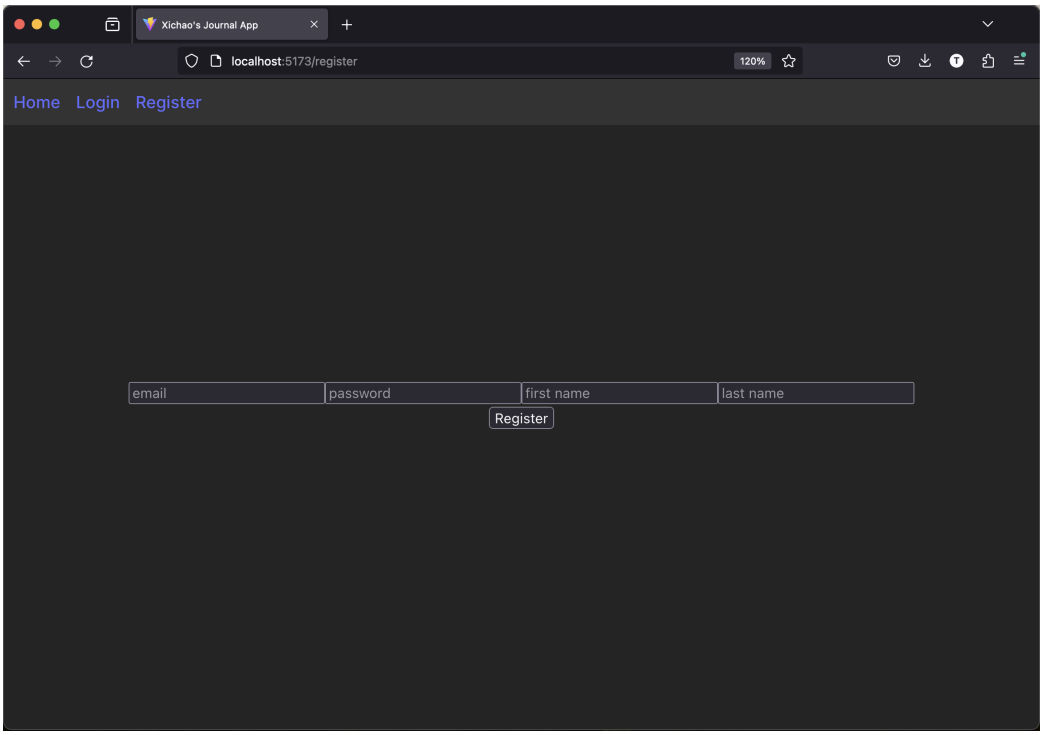


Figure 5.6: Registration page view.

Profile Page

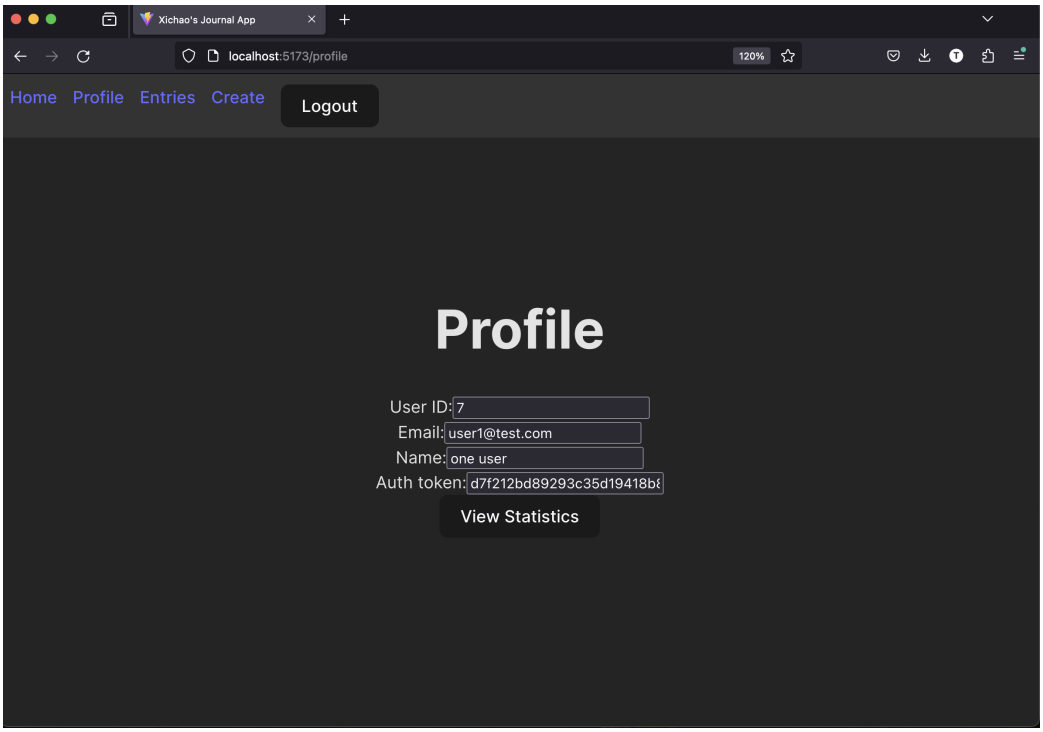


Figure 5.7: User profile page. Displays user information.

Require Login Page

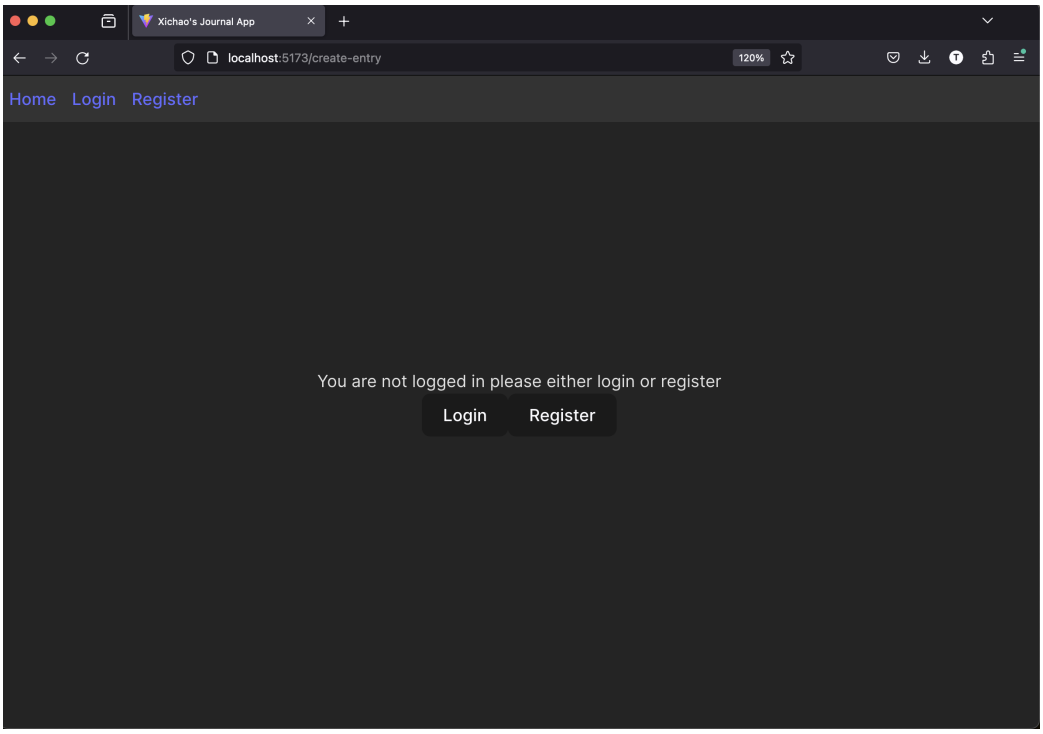


Figure 5.8: Page displayed when accessing pages that require user to be logged in.

Statistics Page

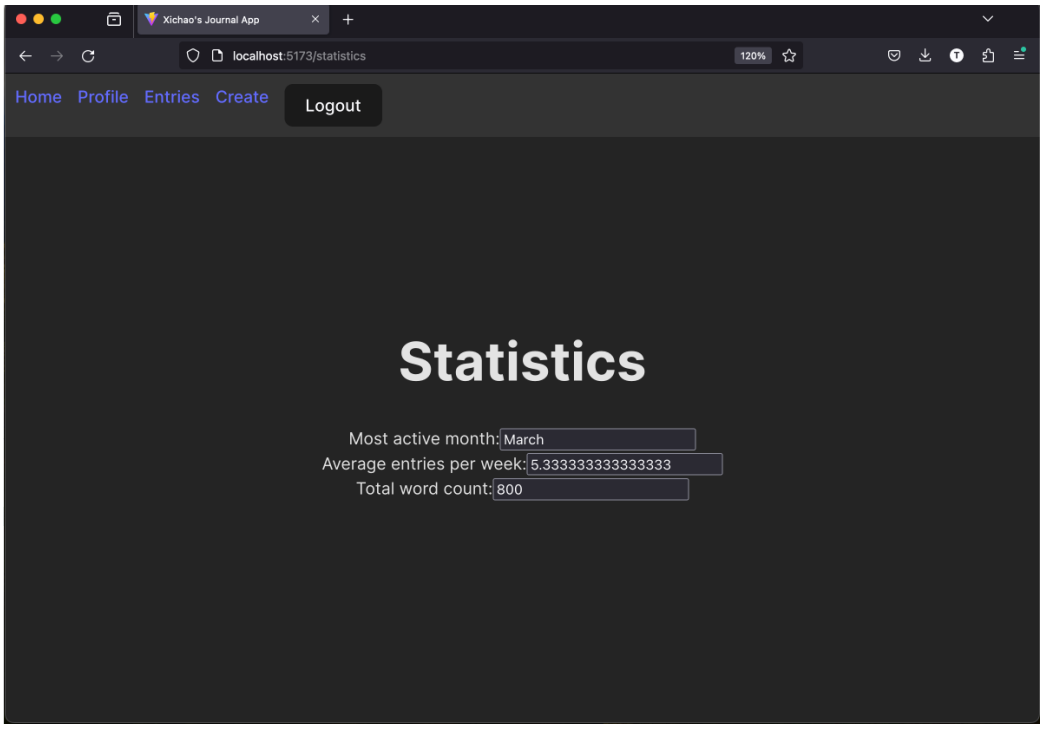


Figure 5.9: Statistics page. Shows statistics about the user's entry pattern.

5.3.3 User Interaction

View Entries

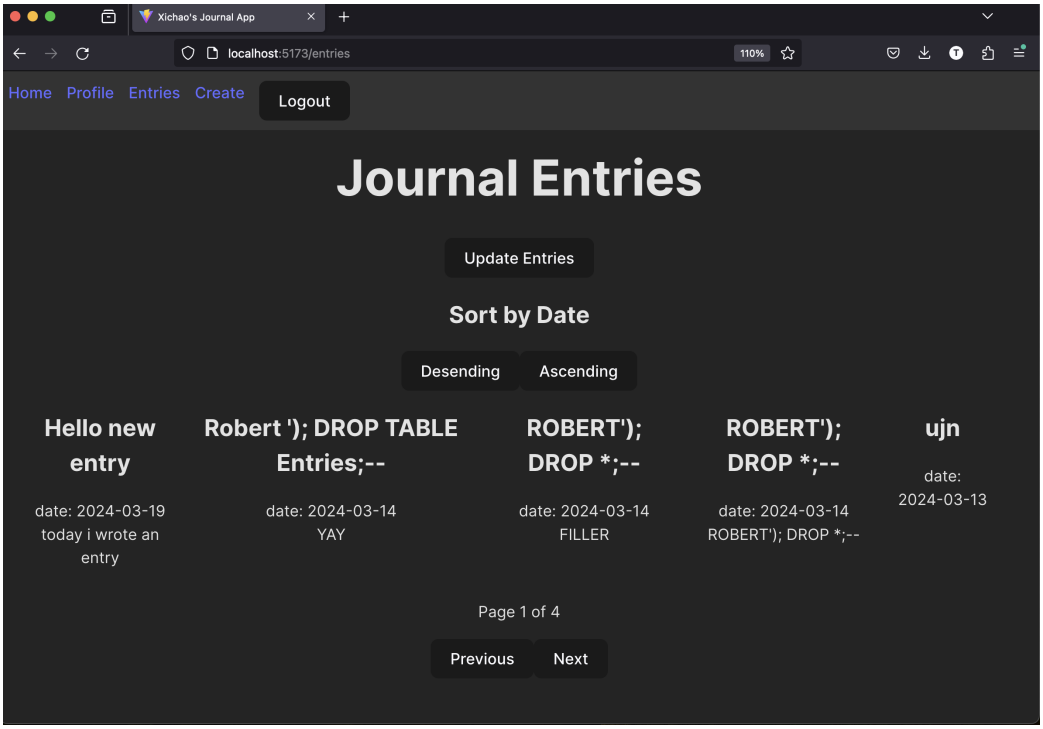


Figure 5.10: View entries page. Displays the user's journal entries in a paginated form, with the option to sort entries.

Create Journal Entry

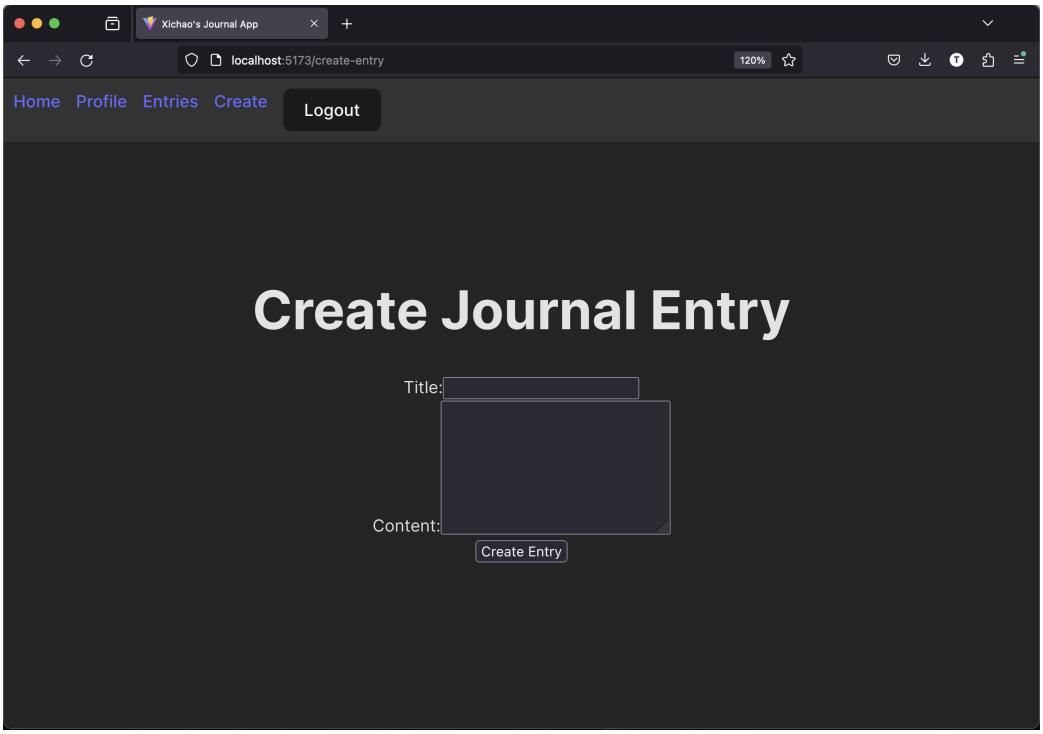


Figure 5.11: Create journal entry page.

5.4 Testing Video

The video demonstrates the app in action, showing the app working and meeting the original requirements. The video can be found at the following link: `https://drive.google.com/drive/folders/1aORi1Iy9CklbfWv4FY6oDLnTgXSax8Gc?usp=sharing`

Figure 5.12: Testing Videos

5.5 System Tests

Comparing my current app against my original requirements:

ID	Requirement Description	Has Been Met
1.1	Design data models for users, and journal entries, with relationships defined.	Yes
1.2	Have an effective way to check the submitted user information, journal entries, and related data before storing it in the database, ensuring the data stored are in the required format.	Yes
1.3	Have a way to generate statistics of the user's journaling habits, such as the most active month of the user, average entries made per week, etc.	Yes
2.1	Develop API endpoints with functionalities implemented to handle user authentication (login and registration) and other core endpoints that drive the journal app as a whole	Yes
2.2	Features for secure password storage(salting and hashing) and and authenticate; token generation for session management.	Yes
3.1	Ensure all API endpoints are secured and accessible only to authenticated users, using tokens or similar mechanisms for session management.	Yes
3.2	Ensure smooth and spontaneous communication with the Frontend.	Yes
4.1	Design the Backend with the room for more functionalities	Yes

Table 5.2: Backend Requirements for Journal App

ID	Requirement Description	Has Been Met
1.1	Develop a login page to authenticate users, include forms for user input and dynamically provide error/feedback when necessary.	Yes
1.2	Implement a registration page enabling new users to create an account, take in input for email, password, last name, and first name, and interact/update Backend. Provide feedback to the user after submission.	Yes
1.3	Develop a place where the user can see details about themselves ie their personal information and journal statistics	Yes
2.1	Create a page for users to compose new journal entries. Input parameters are the title and content of the entry. Interact with the corresponding endpoint and provide feedback to the user.	Yes
2.2	Design a retrieve journal page that lists all the user's entries, with options for sorting the entries in ascending or descending order based on creation date or other criteria in an efficient way.	Yes
3.1	Implement a navigation bar that adjusts its visibility of options based on the user's login status.	Yes
4.1	Utilize state management techniques to keep the user logged in across different pages, preserving session information securely.	Yes
4.2	Handle fetching, posting, and updating data through JSON responses from the Backend.	Yes

Table 5.3: Frontend Requirements for Journal App

Chapter 6

Evaluation

6.1 Requirements Specification Evaluation

From my testing section, it can be confirmed that all the originally proposed requirements from the specification have been fully met.

6.2 Independent End-User Feedback

A very critical friend of mine has tested and provided constructive feedback on the application. Here are the feedbacks:

Positive Feedback

- all the features work as expected of a journal app
- the clean design of the website is a positive aspect
- liked the fact that the website was fast
- He thinks it's easy to use.

- the user liked the fact that the website was secure and the passwords are hashed
- liked how the entries are displayed
- thought the statistics page was a nice touch

Constructive Criticism

- The error message after an API call is too "codey"
  If there is an error after an API call, I conditionally render the Error message by simply parsing it as a string and displaying it. A potential improvement could be creating error pages that display the error message based on the error code.
- The content box for the journal entry is too small
  In the future I could make a better UI for the journal entry page, potentially making my custom reusable input box and button components.
- The user would have liked to see individual journal entries better
- Editing journal entries
- Deleting journal entries
  At the moment the user can only view the journal entries and five entries are displayed on a page. The user would like to see individual journal entries better, edit, and delete them. This could be implemented by adding a button to each journal entry that allows the user to expand on the entry. Then the user can edit or delete the entry. I already have the API endpoints for deleting journal entries, but due to time constraints, I was not able to implement the Frontend for it.
- The user would have liked to go back to the home page after an invalid URL
  He tried to access an invalid URL, and the page was blank. I could implement a 404 page that redirects the user back to the home page.
- The user would like to search by date or search by journal
  A search bar could be implemented that allows the user to search by date or search by journal. This would be a useful design

6.3 Improvements

There are many potential improvements that could be made to the project. Here are some of them:

- Mobile App
  The Backend is already very sophisticated with many endpoints, so a mobile app could be developed that uses the same API endpoints. This would allow users to write journal entries on the go. A mobile app could also have features like push notifications to remind the user to write a journal entry. In addition, a mobile app could allow users to write journals offline and sync when they are online.

- Email
  A feature that I have started implementing is sending emails to users daily to prompt them to write a journal entry. Then if they respond to the email, the email will be saved as a journal entry. This feature could be created via a cron job that runs every day and sends an email to all users who have not written a journal entry in the last 24 hours. This feature could be implemented using the Django Email API.

- Sentiment Analysis
  I have tried making calls to the API provided by Google Cloud Natural Language API, but I haven't fully implemented it. This would be useful information to have providing insight into the user's overall mental health.

- Better Analysis
  The analysis section could be improved by adding more detailed statistics about the user's journal entries. For example, I could analyse the user's sentiment over time to let the user know if they are feeling better or worse. I could also analyse the user's most common words to see what they are writing about the most.

- Photo and Audio Upload
  It would be cool to be able to include pictures or audio. Recently I found some old pictures and it made me feel nostalgic, so having

the ability to include pictures or audio in a journal entry would be a nice feature to have. This could be implemented by allowing the user to upload a picture or audio file and then storing the URL in the database. Then the user could view the picture or listen to the audio in the journal entry.

Chapter 7

Appendix

7.1 Database SQL Tests

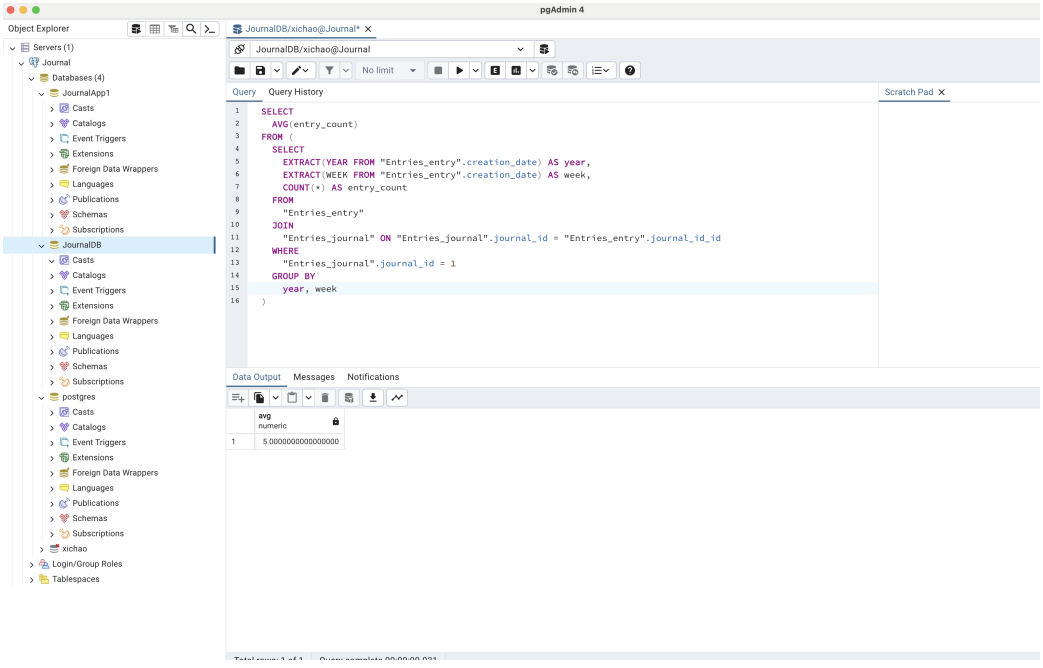


Figure 7.1: Query result for average entry per week.

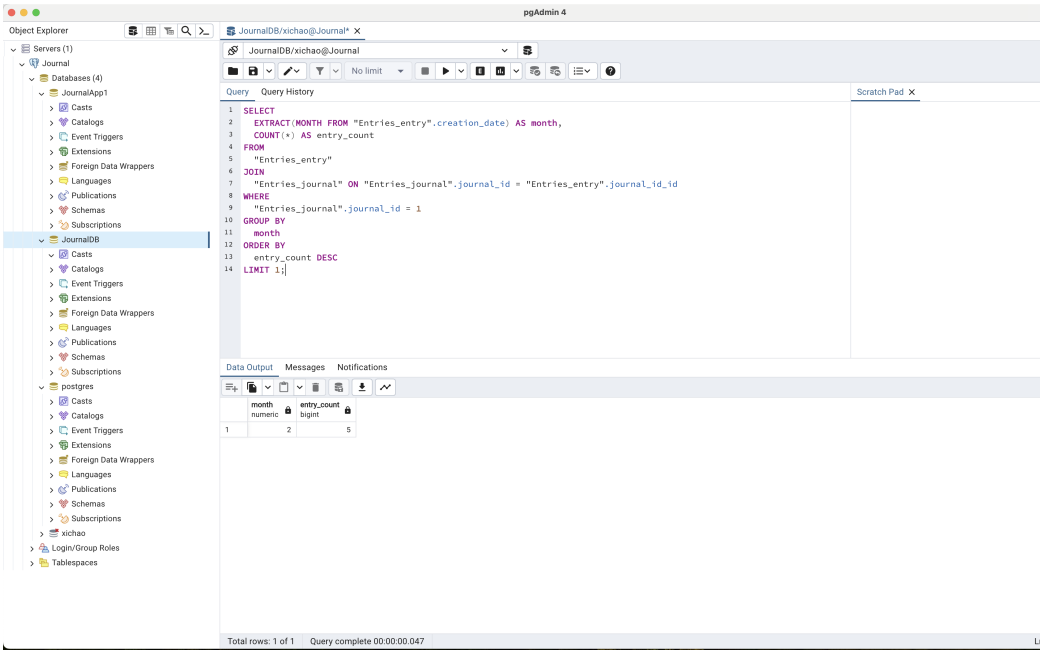


Figure 7.2: Query result for most active month.

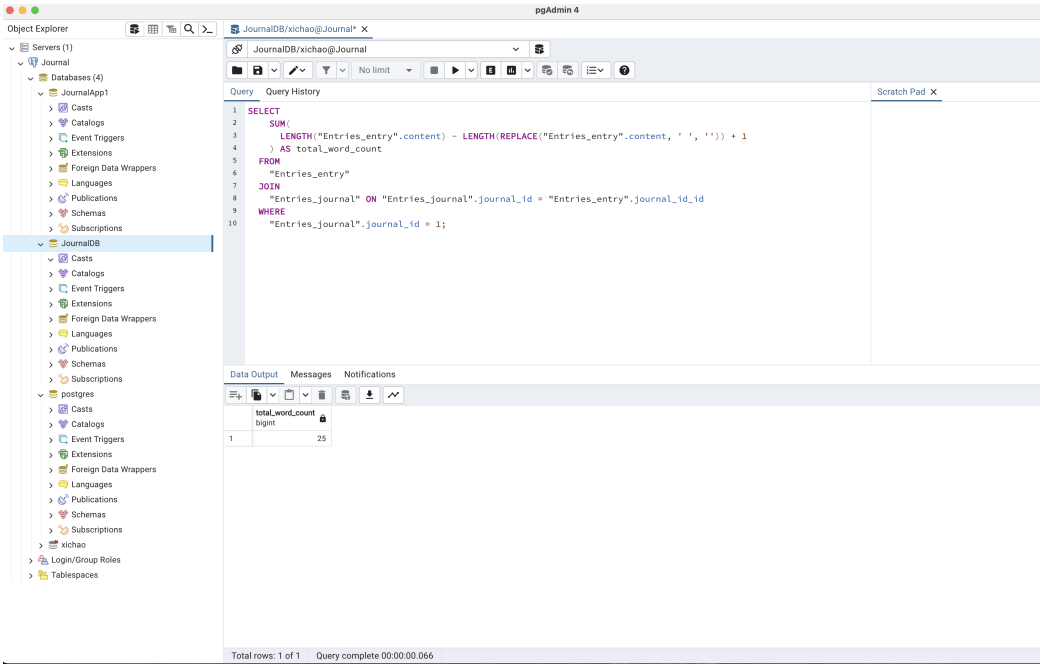


Figure 7.3: Query result for total word count.

7.2 API Test Details

7.2.1 User App

7.2.2 register/

Figure 7.4: Request to register a new user that already exists

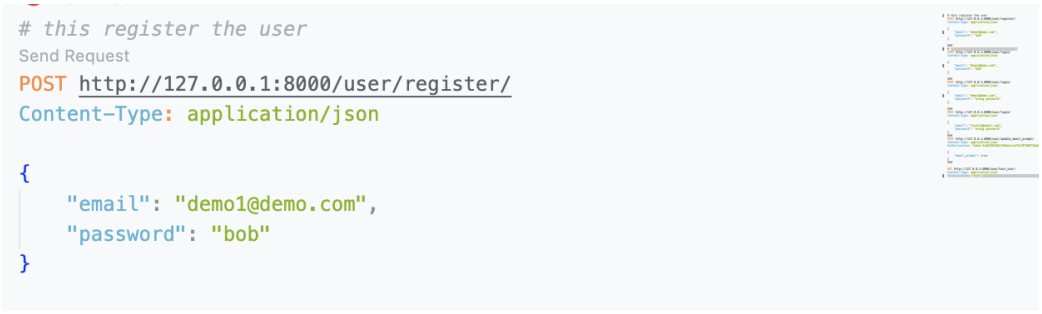
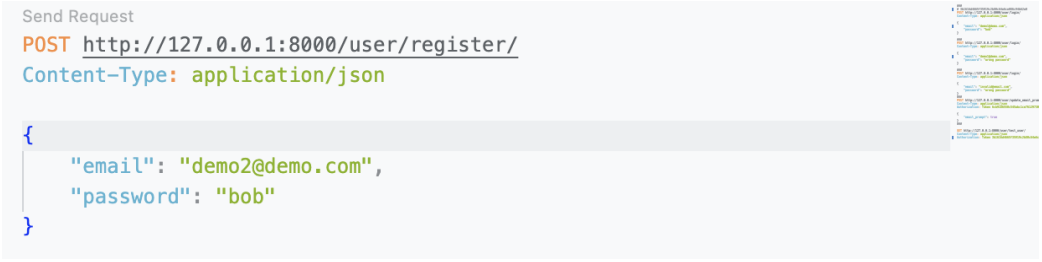


Figure 7.6: Request to register a new user that does not exist



H

Figure 7.5: Response to register a new user that already exists

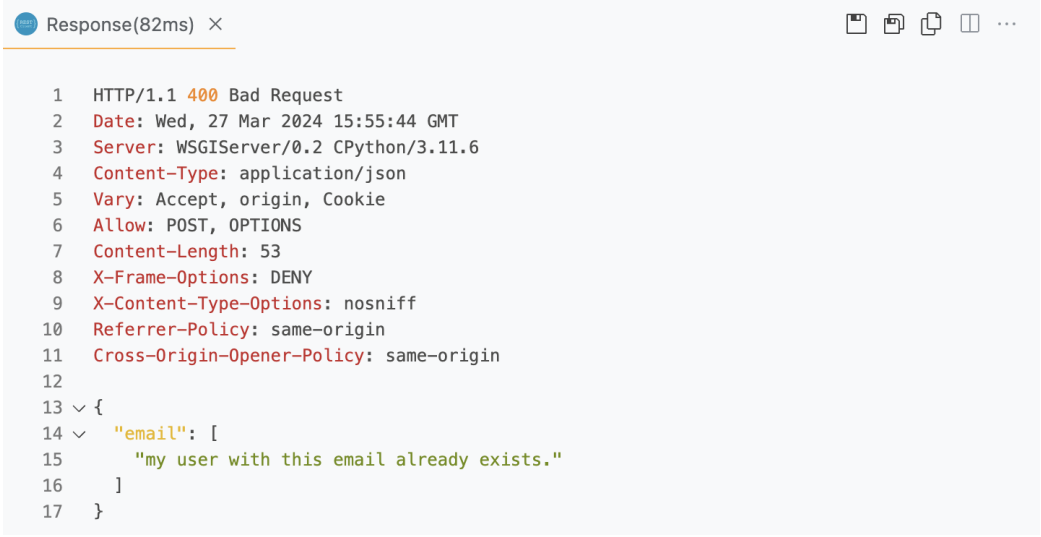
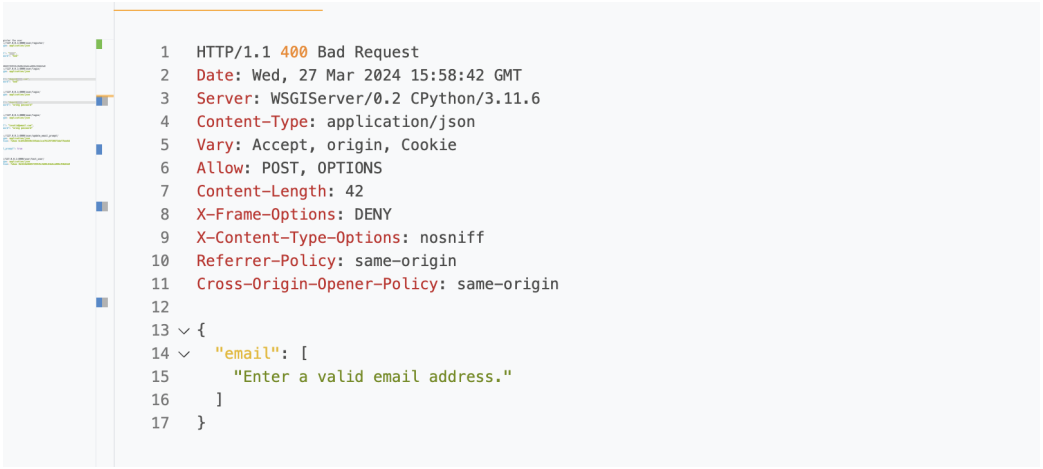


Figure 7.7: Response to register a new user that does not exist

Figure 7.8: Request to register a new user with an invalid email

Figure 7.9: Response to register a new user with an invalid email



7.2.3 login/

Figure 7.10: Request to login with a valid user

```
POST http://127.0.0.1:8000/user/login/
Content-Type: application/json

{
    "email": "demo2@demo.com",
    "password": "bob"
}
```

Figure 7.11: Response to valid login

Response(54ms)

```
HTTP/1.1 200 OK
Date: Wed, 27 Mar 2024 16:00:48 GMT
Server: WSGIServer/0.2 CPython/3.11.6
Content-Type: application/json
Vary: Accept, origin, Cookie
Allow: POST, OPTIONS
Content-Length: 129
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Referrer-Policy: same-origin
Cross-Origin-Opener-Policy: same-origin

{
  "user_id": 25,
  "email": "demo2@demo.com",
  "first_name": null,
  "last_name": null,
  "authToken": "3e6ee5a3567f2abf016d301640a6d453e3d9c1fd"
}
```

Figure 7.12: Request to login with an invalid user

```
POST http://127.0.0.1:8000/user/login/
Content-Type: application/json

{
    "email": "invalid@email.com",
    "password": "wrong password"
}
###
```

Figure 7.13: Response to invalid login with an invalid user

```
Response(44ms)

HTTP/1.1 404 Not Found
Date: Wed, 27 Mar 2024 16:04:24 GMT
Server: WSGIServer/0.2 CPython/3.11.6
Content-Type: application/json
Vary: Accept, origin, Cookie
Allow: POST, OPTIONS
Content-Length: 31
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Referrer-Policy: same-origin
Cross-Origin-Opener-Policy: same-origin

{
  "error": "User does not exist"
}
```

Figure 7.14: Request to login with an invalid password

```
POST http://127.0.0.1:8000/user/login/
Content-Type: application/json

{
    "email": "demo2@demo.com",
    "password": "wrong password"
}
```

Figure 7.15: Response to invalid login with an invalid password

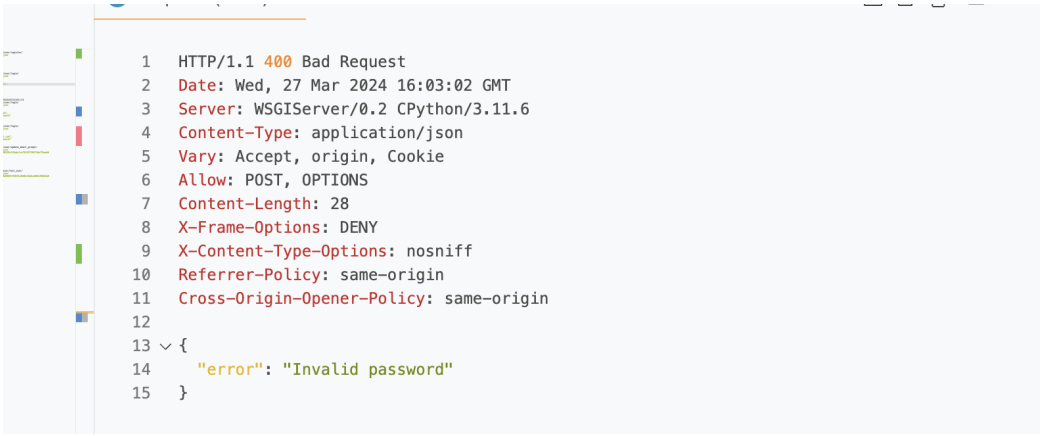


Figure 7.16: Request to login with an invalid email

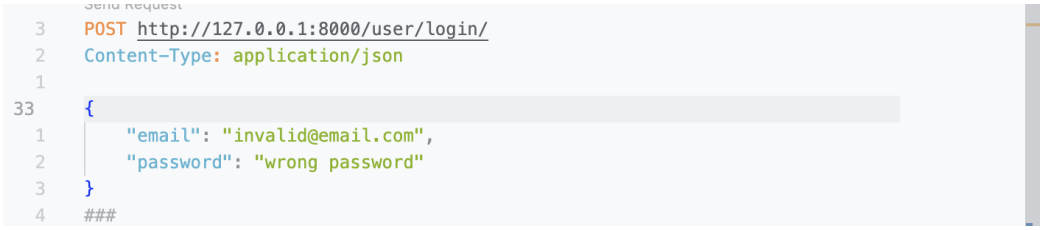


Figure 7.17: Response to invalid login with an invalid email

7.2.4 test_user/

This is used to test the user authentication. The user must be authenticated with a token to access this endpoint. Also it tests wether the server is able to retrieve the user information from the token.

Figure 7.18: Request to test user authentication with a valid token

Figure 7.19: Response to test user authentication with a valid token

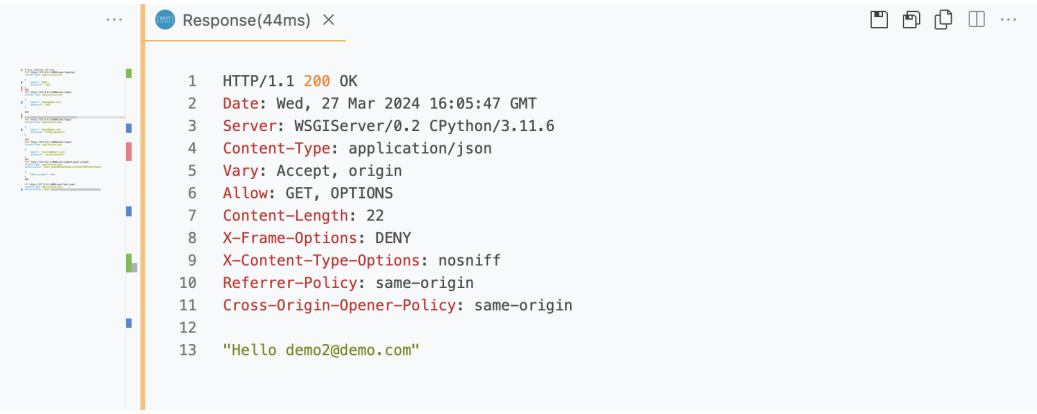
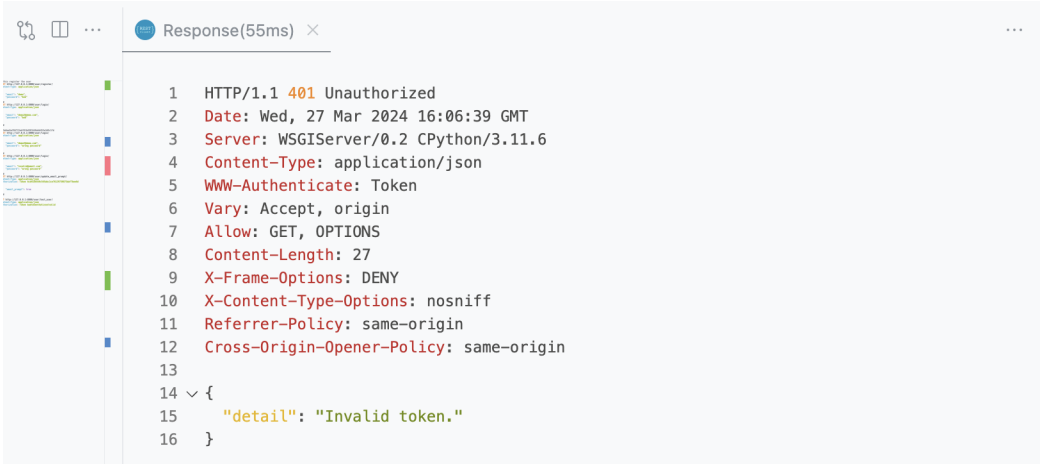


Figure 7.20: Request to test user authentication with an invalid token

Figure 7.21: Response to test user authentication with an invalid token

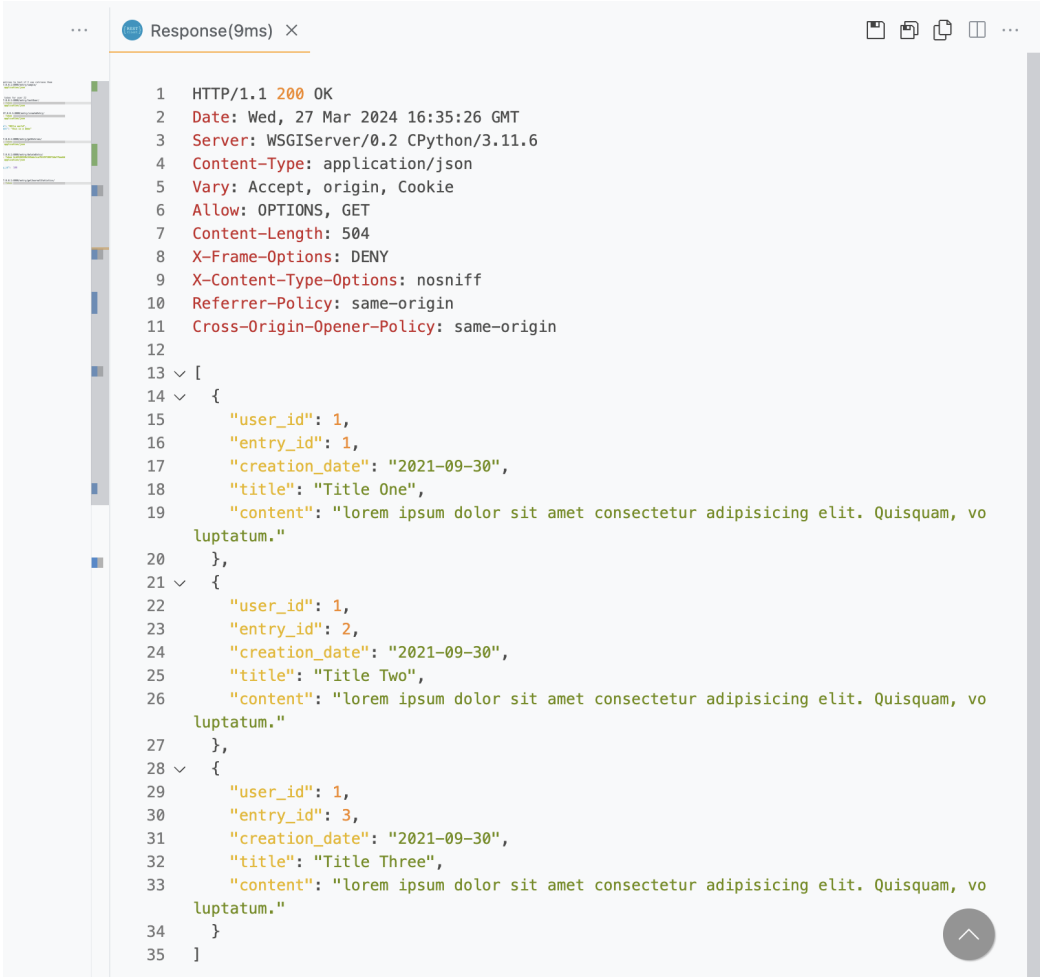


7.2.5 Entry App

7.2.6 sample/

Figure 7.22: Request to get samples

Figure 7.23: Response to get samples



7.2.7 testUser/

Figure 7.24: Request to test user in entry app

Figure 7.25: Response to test user in entry app

7.2.8 createEntry/

Figure 7.26: Request to create a new entry

Figure 7.27: Response to creating a new entry

7.2.9 getEntries/

Figure 7.28: Request to get an entry

```
Send Request
GET http://127.0.0.1:8000/entry/getEntries/
Authorization: Token 3b1616d4665f35919c2b80c64e6ce086c94b62e0
Content-Type: application/json
```

Figure 7.29: Response to get an entry

Response(97ms)

```
HTTP/1.1 200 OK
Date: Wed, 27 Mar 2024 16:40:28 GMT
Server: WSGIServer/0.2 CPython/3.11.6
Content-Type: application/json
Vary: Accept, origin
Allow: OPTIONS, GET
Content-Length: 213
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Referrer-Policy: same-origin
Cross-Origin-Opener-Policy: same-origin

[
  {
    "user_id": 22,
    "entry_id": 57,
    "creation_date": "2024-03-26",
    "title": "HElo world",
    "content": "this is a demo"
  },
  {
    "user_id": 22,
    "entry_id": 71,
    "creation_date": "2024-03-27",
    "title": "new entry",
    "content": "this is a test"
  }
]
```

7.2.10 deleteEntry/

Figure 7.30: Request to delete an entry with an invalid id

Figure 7.31: Response to invalid delete entry request with invalid id

Figure 7.32: Request to delete an entry with valid id

Figure 7.33: Response to valid delete entry request with valid id

7.2.11 getJournalStatistics/

Figure 7.34: Request to get journal statistics

Figure 7.35: Response to get journal statistics

7.3 Git Log

```
* commit 4d1a3bb (HEAD -> main, origin/main, origin/HEAD)
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   4 hours ago
```

```
|
|     commented sorting implementation
|
* commit 3a8f8b6
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   5 hours ago
|
|     implemented timsort
|
* commit f2c5d9d
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   2 days ago
|
|     commenting:
|     for files involved with use context
|     api calling
|
* commit af1889f
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   6 days ago
|
|     add statistics page
|
* commit 8d0d6ab
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 days ago
|
|     removed unnecssary files
|
* commit 95028b1
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 days ago
|
|     Now error response is being displayed
|
* commit 2321eeb
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 days ago
|
|     update:
|     - display prompt to sign in if user tries to make entries or view
↪  entries without logged in
|
* commit c3d4df9
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
```

```
| Date:   13 days ago
|
|     updated Entries page:
|     - fixed logic error, now sort sorts all items instead of just
↪  current items
|
* commit e3a64b3
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   2 weeks ago
|
|     retrieve Journal Pagination and fixed logic error
|
* commit 7b2bbac
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   2 weeks ago
|
|     update retrieve entries page:
|     - uses local storage
|     - sorting by date
|     - sorting function
|
* commit 629ce66
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     fixed create entry logic such that it display errors
|
* commit 7712f72
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     updated README
|
* commit 42f601b
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     Update README.md
|
* commit 3fb1621
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     - updated Entry so that it includes dates.
|     - updated cookie policy.
```

```
|
* commit 3e4b195
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     working profile that allow user to opt in for journal entry
|
* commit fbb84c2
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     - working create entry page
|
* commit 7d74e08
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     - updated Navbar to work with contex
|     - implementation of interceptor (not sure if it works but it doesnt
↪  do anything bad)
|     - register page updated to useAuth (context)
|     - fixed bugs: JournalEntriesPage
|
* commit f8460d7
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     stored user in context!!
|
* commit 32ffa4e
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     Working profile page
|     Journal Entry page able to fetch sample data but user doesnt work
|
|     committing before trying to dabble with context again.
|
* commit 1fd04c5
  Author: Xichao <49397940+TheXichao@users.noreply.github.com>
  Date:   7 weeks ago

      initial commit: had to revert back to a working version then made
      ↪  some updates from there.

```

```
135
136
137 * commit da2f097 (HEAD -> main)
138 | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
139 | Date:   26 hours ago
140 |
141 |     test created for the Entries app
142 |
143 * commit fa077bc (origin/main, origin/HEAD)
144 | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
145 | Date:   2 days ago
146 |
147 |     rate limit
148 |     user/test.py
149 |
150 * commit ef5c594
151 | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
152 | Date:   3 days ago
153 |
154 |     models urls ect
155 |
156 * commit 98ba6f6
157 | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
158 | Date:   6 days ago
159 |
160 |     statistics api + tests
161 |
162 * commit 5961d98
163 | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
164 | Date:   7 days ago
165 |
166 |     Delete Entry Endpoint and test
167 |
168 * commit 3fefe5c
169 | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
170 | Date:   8 days ago
171 |
172 |     - secret.json
173 |     removal of
174 |     -unnecessary API class
175 |     - google sentiment fetching
176 |
177 * commit 043d491
178 | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
179 | Date:   13 days ago
```

```
|
|     - new ER diagram
|     - edited entries class so that data can be edited from admin panel
|
* commit 10a3bae
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   3 weeks ago
|
|     bug fix + begin email implementation and cronjob stuff
|
* commit 2a85b3c
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     added journal entry boolean to user table.
|     created endpoint to modify the journal_prompt feature
|
* commit 32a4385
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     -minor bug change
|
* commit f0c111f
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     working create and get all entry end-points
|
* commit 12403a3
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     Create Journal upon new User creation using signals
|
* commit d8b0542
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     Moved react frontend to a different git Repo
|
* commit 30efcb7
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
```

```
|     I am defeated. Attempted to use context for user authentication
↪  however nothing works as intended. The code in this commit doenst
↪  fully work. I might revert later...
|
* commit 6a59483
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     Refactored RegisterPage so that it uses an interface and one single
↪  state to represent all the RegisterRequestData by using typescript
↪  interface. Neat.
|
* commit 5b170ee
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   7 weeks ago
|
|     Currently working code for user sign up and login.
|
* commit e646360
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   8 weeks ago
|
|     superdoper all purpose api call function
|
*   commit 3026d23
|\  Merge: f187a4d 2983cef
| | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| | Date:   8 weeks ago
| |
| |     Merge pull request #1 from
↪  TheXichao/UI-homepage;-displaying-and-creating-journal
| |
| |     Mergin my frontend progress so far with the main branch
| |
| * commit 2983cef
| | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| | Date:   8 weeks ago
| |
| |     begin creating register form
| |
| * commit aa0040f
| | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| | Date:   8 weeks ago
| |
| |     Created working login form with working login functionality.
↪  Returned cookies are also stored in cookies.
```

```
| |
| * commit b6f193d (origin/frontend-development)
| | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| | Date:   8 weeks ago
| |
| |     login form implementation using react-hook-form
| |
| * commit aa1cb85
| | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| | Date:   8 weeks ago
| |
| |     working post request, able to login, render test user info in
↪  login page
| |
| * commit 4b2b076
| | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| | Date:   8 weeks ago
| |
| |     discovered and fixed the error that prevented communication
↪  between my backend and frontend. fixed CORS error that enabled
↪  communciation on both sides. For the django site I added a CORS
↪  middleware and for vite I added a proxy.
| |
| * commit fc0cdfe
| | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| | Date:   8 weeks ago
| |
| |     ready Navbar
| |
| * commit 2cd956d
| | Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| | Date:   8 weeks ago
| |
| |     react pages to be created
| |
| * commit 47de3cb
|/  Author: Xichao <49397940+TheXichao@users.noreply.github.com>
|   Date:   8 weeks ago
|
|       created status bar and the corresponding pages
|
* commit f187a4d
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   8 weeks ago
|
```

```
|     - use axios to fetch data
|     - created hook to make get request
|     - use my own hook in the app
|
* commit 2734642
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   8 weeks ago
|
|     - created react frontend (vite)
|     - tried fetching data using fetch
|     - learned to useState and useRef
|     - Async
|     - understanding type script
|     - interface
|     - conditional rendering
|
|     Todo:
|     - use axios
|     - learn post
|     - authenticated requests
|     - cookies
|
* commit ee10bad
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   8 weeks ago
|
|     Created views to be implemented to fetch journals ect. Fixed bugs.
|
* commit 96de4a3
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   9 weeks ago
|
|     Authentication working, allowing user to login and verity user
|
* commit 56c13c5
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   9 weeks ago
|
|     functioning login and logout methods which returns valid token
|
* commit 84c33be
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   9 weeks ago
|
|     Fixed bugs
```

```
|
* commit a9e4ccf
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   9 weeks ago
|
|     Test and beginning working on user authentication using token based
↪  authentication using REST framework
|
|     fixed minor bugs
|
* commit 90793d8
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   9 weeks ago
|
|     changed to local postgre db
|     customised UserPannel
|     Fixed Superuser permission
|     added appropriate URL patterns for users
|
* commit 1437e00
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   9 weeks ago
|
|     test for MyUser,
|     fixed hashing with salt,
|     updated MyUser,
|     Registered Models for Admin
|
* commit eb3b58e
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   9 weeks ago
|
|     Basic user encryption and authentication implementation. To be
↪  fixed and completed!!
|
* commit 06fa6d2
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   10 weeks ago
|
|     removed redundency
|
* commit 895fd35
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   10 weeks ago
|
```

```
|     initial implementation of functionalities for MyUser
|
* commit 5d6963e
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   10 weeks ago
|
|     added requirements
|
* commit 0f8eadf
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   10 weeks ago
|
|     update ER diagram logic
|
* commit babe810
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   10 weeks ago
|
|     update models
|
* commit 59a5a0d
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   10 weeks ago
|
|     updated MyUser Models with methods to be implemented
|
* commit 048f65c
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   10 weeks ago
|
|     created the django app responsible for API endpoints using the
↪  DjangoRestframework
|
* commit b2e3744
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   2 months ago
|
|     fixed diagram
|
* commit 1f174f8
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   2 months ago
|
|     database diagram edit
|
```

```
* commit 7d05e47
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   2 months ago
|
|     Fixed grammar in my database diagram and the name of my models to
↪  follow the convention name for a django model. Specifically changing
↪  from plural to singular for model names.
|
* commit 92287e6
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   2 months ago
|
|     first attempt at implementing the database models for the app
|
* commit 24e1821
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   3 months ago
|
|     .gitignore file
|
* commit 145e741
| Author: Xichao <49397940+TheXichao@users.noreply.github.com>
| Date:   3 months ago
|
|     Create README.md
|
* commit c14443a
  Author: Xichao <49397940+TheXichao@users.noreply.github.com>
  Date:   3 months ago

      initial set up and commit: created the django template and added
      ↪  the .gitignore file for react&django&macos
```

7.4 Code Listings

/backend/JournalApp/settings.py

```
"""
Django settings for JournalApp project.

Generated by 'django-admin startproject' using Django 4.2.7.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""

from pathlib import Path
from pyparsing import C

import json


# reading the secret key from the secret.json file
def get_secret(secret_name: str) -> str:
    try:
        with open("secret.json") as f:
            secret = json.loads(f.read())
        return secret[secret_name]
    except Exception as e:
        print(e)
        return ""


# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = get_secret("DJANGO_SECRET_KEY")
```

```
39
40  # SECURITY WARNING: don't run with debug turned on in production!
41  DEBUG = True
42
43  ALLOWED_HOSTS = []
44
45
46  # Application definition
47  INSTALLED_APPS = [
48      "django.contrib.admin",
49      "django.contrib.auth",
50      "django.contrib.contenttypes",
51      "django.contrib.sessions",
52      "django.contrib.messages",
53      "django.contrib.staticfiles",
54      # REST framework
55      "rest_framework",
56      "rest_framework.authtoken",
57      # My apps
58      "Users",
59      "Entries",
60      "Email",
61      # Third party apps
62      "django_extensions",
63      "corsheaders",
64      "django_crontab",
65  ]
66
67
68  MIDDLEWARE = [
69      "django.middleware.security.SecurityMiddleware",
70      "django.contrib.sessions.middleware.SessionMiddleware",
71      "django.middleware.common.CommonMiddleware",
72      "django.middleware.csrf.CsrfViewMiddleware",
73      "django.contrib.auth.middleware.AuthenticationMiddleware",
74      "django.contrib.messages.middleware.MessageMiddleware",
75      "django.middleware.clickjacking.XFrameOptionsMiddleware",
76      # middleware for CORS
77      "corsheaders.middleware.CorsMiddleware",
78      "django.middleware.common.CommonMiddleware",
79  ]
80
81  REST_FRAMEWORK = {
82      "DEFAULT_AUTHENTICATION_CLASSES": [
83          "rest_framework.authentication.BasicAuthentication",
```

```
        "rest_framework.authentication.SessionAuthentication",
    ],
    # "DEFAULT_RENDERER_CLASSES": [
    #     "rest_framework.renderers.JSONRenderer",
    # ],
}
ROOT_URLCONF = "JournalApp.urls"

TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS": [],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.messages",
            ],
        },
    },
]

# registering my user to be used for the framework
AUTH_USER_MODEL = "Users.MyUser"

WSGI_APPLICATION = "JournalApp.wsgi.application"


# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    "default": {
        #'ENGINE': 'django.db.backends.sqlite3',
        "ENGINE": "django.db.backends.postgresql",
        "NAME": "JournalDB",
        "USER": "admin",
        "PASSWORD": "admin",
        # "HOST": "roundhouse.proxy.rlwy.net",
        # "PORT": "34716",
    }
}

```

```
129
130 # Password validation
131 #
    ↪ https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators
132
133 AUTH_PASSWORD_VALIDATORS = [
134     {
135         "NAME":
            ↪ "django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
136     },
137     {
138         "NAME":
            ↪ "django.contrib.auth.password_validation.MinimumLengthValidator",
139     },
140     {
141         "NAME":
            ↪ "django.contrib.auth.password_validation.CommonPasswordValidator",
142     },
143     {
144         "NAME":
            ↪ "django.contrib.auth.password_validation.NumericPasswordValidator",
145     },
146 ]
147
148
149 # Internationalization
150 # https://docs.djangoproject.com/en/4.2/topics/i18n/
151
152 LANGUAGE_CODE = "en-us"
153
154 TIME_ZONE = "GB"
155
156 USE_I18N = True
157
158 USE_TZ = True
159
160
161 # Static files (CSS, JavaScript, Images)
162 # https://docs.djangoproject.com/en/4.2/howto/static-files/
163
164 STATIC_URL = "static/"
165
166 # Default primary key field type
167 # https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field
168
```

```
DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"

CORS_ALLOWED_ORIGINS = [
    "http://127.0.0.1:5173",
    "http://localhost:5173",
    "http://172.28.4.231:5173",
]


# https://crontab.guru/
CRONJOBS = [
    ("* * * * *", "Email.cron.send_scheduled_email"),
    ("* * * * *", "Email.cron.daily_scheduled_email"),
]


# Email settings
EMAIL_BACKEND = "django.core.mail.backends.smtp.EmailBackend"
EMAIL_HOST = "smtp.gmail.com"
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = get_secret("EMAIL_HOST_USER")
EMAIL_HOST_PASSWORD = get_secret("EMAIL_HOST_PASSWORD")
```

/backend/JournalApp/urls.py

```
"""
URL configuration for JournalApp project.

The `urlpatterns` list routes URLs to views. For more information please
    see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
```

```
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.shortcuts import render
from django.urls import path, include

urlpatterns = [
    path("admin/", admin.site.urls),
    path("user/", include("Users.urls")),
    path("entry/", include("Entries.urls")),
]
```

/backend/Users/admin.py

```
from django.contrib import admin

# Register your models here.
from .models import MyUser


class MyUserAdmin(admin.ModelAdmin):
    list_display = ("user_id", "email", "first_name", "last_name", "is_staff")


admin.site.register(MyUser, MyUserAdmin)
```

/backend/Users/apps.py

```
from django.apps import AppConfig


class UsersConfig(AppConfig):
    default_auto_field = "django.db.models.BigAutoField"
    name = "Users"

    def ready(self):
        import Users.signals
```

/backend/Users/models.py

```
from distutils.command import clean
from django.db.models.signals import post_save
from django.db import models
from django.contrib.auth.models import AbstractBaseUser, UserManager
from django.contrib.auth.models import PermissionsMixin

from .utils import *

# Create your models here.


def check_email_and_password(email: str, password: str | None) -> bool:
    if password is None:
        raise ValueError("Password is not valid")
    if not validate_email(email):
        raise ValueError("Email is not valid")
    return True


class MyUserManager(UserManager):
    def create_user(self, email, password=None, **extra_fields):
        cleaned_email = clean_email(email)
        is_valid = check_email_and_password(cleaned_email, password)
```

```
        if is_valid:
            user = self.model(email=cleaned_email, **extra_fields)
            user.set_password(password)
            user.save()

    def create_superuser(self, email, password=None, **extra_fields):
        cleaned_email = clean_email(email)
        is_valid = check_email_and_password(cleaned_email, password)
        if is_valid:
            user = self.model(email=cleaned_email, **extra_fields)
            user.set_password(password)
            user.is_staff = True
            user.is_superuser = True
            user.save()


class MyUser(AbstractBaseUser, PermissionsMixin):
    """A Users table used to store the data of my user a subclass
    ↪  implementing the default User model"""

    user_id = models.AutoField(primary_key=True)
    email = models.EmailField(unique=True)
    is_staff = models.BooleanField(default=False)
    creation_date = models.DateField(auto_now_add=True)
    first_name = models.CharField(max_length=30, null=True)
    last_name = models.CharField(max_length=30, null=True)
    email_prompt = models.BooleanField(default=False, null=True)

    # By convention, the manager attribute are named objects.
    objects = MyUserManager()

    USERNAME_FIELD = "email"
    REQUIRED_FIELDS = ["first_name", "last_name"]

    def get_full_name(self) -> str:
        return f"{self.first_name} {self.last_name}"

    def set_password(self, raw_password: str) -> None:
        self.password = hash_password(raw_password)

    def check_password(self, raw_password: str) -> bool:
        return verify_password(raw_password, self.password)


# @receiver(post_save, sender=MyUser)
```

```
# def create_user_journal(sender, instance, created, **kwargs):
#     if created:
#         Journal.objects.create(user=instance)
```

/backend/Users/serializers.py

```
from rest_framework import serializers
from Users.models import MyUser


class MyUserSerializer(serializers.ModelSerializer):
    class Meta(object):
        model = MyUser
        fields = [
            "user_id",
            "first_name",
            "last_name",
            "email",
            "password",
        ]
```

/backend/Users/signals.py

```
from .models import MyUser, MyUserManager
from Entries.models import Journal

from django.dispatch import receiver
from django.db.models.signals import post_save


@receiver(post_save, sender=MyUser)
def create_journal(sender, instance, created, **kwargs):
```

```
    if created:
        Journal.objects.create(user_id=instance)
        instance.journal.save()
```

/backend/Users/urls.py

```
from . import views
from django.urls import re_path

#
urlpatterns = [
    re_path("register/", views.register_view, name="api_register"),
    re_path("login/", views.login_view, name="api_login"),
    re_path("logout/", views.logout_view, name="api_logout"),
    re_path("test_user/", views.test_user, name="test_user"),
    re_path("update_email_prompt/", views.update_email_prompt,
    ↪  name="email_prompt"),
    # path("session/", views.session_view, name="api_session"),
]
```

/backend/Users/utils.py

```
import hashlib
import os
import re


# def hash_content(content: str) -> str:
#     sha256_hash = hashlib.sha256()
#     sha256_hash.update(content.encode("utf-8"))
#     return sha256_hash.hexdigest()

```

```

def hash_password(password: str) -> str:
    salt = os.urandom(16)
    salted_password = salt + password.encode("utf-8")

    sha256_hash = hashlib.sha256()
    sha256_hash.update(salted_password)
    hashed_password = sha256_hash.hexdigest()

    return f"{salt.hex()}:{hashed_password}"


def verify_password(password: str, hashed_password: str) -> bool:
    salt, stored_hash = hashed_password.split(":")

    salted_password = bytes.fromhex(salt) + password.encode("utf-8")
    sha256_hash = hashlib.sha256()
    sha256_hash.update(salted_password)
    provided_hash = sha256_hash.hexdigest()

    return provided_hash == stored_hash


def validate_email(email: str) -> bool:
    return re.match(r"[^@]+@[^@]+\.[^@]+", email) is not None


def clean_email(email: str) -> str:
    cleaned_email = re.search(r"[^@]+@[^@]+\.[^@]+", email)
    if cleaned_email is not None:
        return cleaned_email.group(0)
    else:
        return ""
```

/backend/Users/views.py

```
from operator import is_
from re import M
```

```
import token
from rest_framework.response import Response
from rest_framework.decorators import api_view
from rest_framework import status
from rest_framework.authtoken.models import Token

from rest_framework.permissions import IsAuthenticated
from rest_framework.decorators import authentication_classes,
↪  permission_classes
from rest_framework.authentication import TokenAuthentication,
↪  SessionAuthentication

from .serializers import MyUserSerializer
from .models import MyUser


# my user views
# TODO: improve security, secure password in response, etc


@api_view(["POST"])
def register_view(request) -> Response:
    serializer = MyUserSerializer(data=request.data)
    if serializer.is_valid():
        user = serializer.save()
        user = MyUser.objects.get(email=request.data["email"])
        user.set_password(request.data["password"])
        user.save()
        token = Token.objects.create(user=user)
        response_data = {
            "user_id": user.user_id,
            "email": user.email,
            "first_name": user.first_name,
            "last_name": user.last_name,
            "authToken": token.key,
            "email_prompt": user.email_prompt,
        }
        return Response(
            response_data,
            status=status.HTTP_201_CREATED,
        )

    return Response(serializer.errors,
    ↪  status=status.HTTP_400_BAD_REQUEST)

```

```
45
46  @api_view(["POST"])
47  def login_view(request) -> Response:
48      try:
49          user = MyUser.objects.get(email=request.data["email"])
50          if user.check_password(request.data["password"]):
51              token, created = Token.objects.get_or_create(user=user)
52              user_data = MyUserSerializer(user).data
53              user_data["token"] = token.key
54              response_data = {
55                  "user_id": user.user_id,
56                  "email": user.email,
57                  "first_name": user.first_name,
58                  "last_name": user.last_name,
59                  "authToken": token.key,
60              }
61              return Response(response_data, status=status.HTTP_200_OK)
62          else:
63              return Response(
64                  {"error": "Invalid password"},
                    ↪  status=status.HTTP_400_BAD_REQUEST
65              )
66
67      except MyUser.DoesNotExist:
68          return Response(
69              {"error": "User does not exist"},
                ↪  status=status.HTTP_404_NOT_FOUND
70          )
71
72
73  @api_view(["POST"])
74  def logout_view(request) -> Response:
75      return Response({})
76
77
78  @api_view(["GET"])
79  @authentication_classes([TokenAuthentication, SessionAuthentication])
80  @permission_classes([IsAuthenticated])
81  def test_user(request) -> Response:
82      return Response(f"Hello {request.user.email}")
83
84
85  @api_view(["POST"])
86  @authentication_classes([TokenAuthentication, SessionAuthentication])
87  @permission_classes([IsAuthenticated])
```

```
def update_email_prompt(request) -> Response:
    user = request.user
    user.email_prompt = request.data["email_prompt"]
    user.save()
    return Response(
        {"message": f"Prompt updated successfully to
        ↪  {user.email_prompt}"},
        status=status.HTTP_201_CREATED,
    )
```

/backend/Entries/admin.py

```
from django.contrib import admin

# Register your models here.
from .models import Journal, Entry

admin.site.register(Journal)
admin.site.register(Entry)
```

/backend/Entries/apps.py

```
from django.apps import AppConfig


class EntriesConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'Entries'
```

/backend/Entries/models.py

```
from email.policy import default
from django.db import models
from Users.models import MyUser
from django.utils import timezone
from django.db import connection

# sentiment analysis
from google.cloud import language_v2


# Create your models here.

"""
The table I have are Users, Journals, and Entries,where they are all linked together.
Each user should have on Journal (foreign key pointing to users),
Each Journal should have multiple entries (foreign key pointing to Journals)
"""


class Journal(models.Model):
    """A Journals table pointing to the owner used to link entries together with the user"""

    journal_id = models.AutoField(primary_key=True)
    user_id = models.OneToOneField(MyUser, on_delete=models.CASCADE)
    creation_date = models.DateField(auto_now_add=True)

    def get_all_entries(self):
        return self.entry_set.all()

    def get_most_active_month(self):
        with connection.cursor() as cursor:
            cursor.execute(
                f"""
SELECT
  EXTRACT(MONTH FROM "Entries_entry".creation_date) AS month,
  COUNT(*) AS entry_count
FROM
  "Entries_entry"
```

```
JOIN
  "Entries_journal" ON "Entries_journal".journal_id =
  ↪  "Entries_entry".journal_id_id
WHERE
  "Entries_journal".journal_id = {self.journal_id}
GROUP BY
  month
ORDER BY
  entry_count DESC
LIMIT 1;
                """
            )
            row = cursor.fetchone()
            return row[0] if row else None

    def get_average_entries_per_week(self):
        with connection.cursor() as cursor:
            cursor.execute(
                f"""
SELECT
  AVG(entry_count) AS avg_weekly_entries
FROM (
  SELECT
    EXTRACT(YEAR FROM "Entries_entry".creation_date) AS year,
    EXTRACT(WEEK FROM "Entries_entry".creation_date) AS week,
    COUNT(*) AS entry_count
  FROM
    "Entries_entry"
  JOIN
    "Entries_journal" ON "Entries_journal".journal_id =
    ↪  "Entries_entry".journal_id_id
  WHERE
    "Entries_journal".journal_id = {self.journal_id}
  GROUP BY
    year, week
) AS weekly_counts;
                """
            )
            row = cursor.fetchone()
            return row[0] if row else None

    def get_total_word_count(self):
        with connection.cursor() as cursor:
            cursor.execute(
                f"""
```

```
SELECT
  SUM(
    LENGTH("Entries_entry".content) -
    ↪  LENGTH(REPLACE("Entries_entry".content, ' ', '')) + 1
  ) AS total_word_count
FROM
  "Entries_entry"
JOIN
  "Entries_journal" ON "Entries_journal".journal_id =
  ↪  "Entries_entry".journal_id_id
WHERE
  "Entries_journal".journal_id = {self.journal_id};
                """
            )
            row = cursor.fetchone()
            return row[0] if row else None

    def __str__(self):
        return f"{self.user_id}'s Journal"


class Entry(models.Model):
    """An Entry table used to store the data of t journal entry and also
    ↪  tell us which journal it belongs to"""

    entry_id = models.AutoField(primary_key=True)
    journal_id = models.ForeignKey("Journal", on_delete=models.CASCADE)
    creation_date = models.DateField(default=timezone.now, editable=True)
    title = models.CharField(max_length=50, default="Untitled")
    content = models.TextField()
    sentiment = models.FloatField(default=None, blank=True, null=True)

    def __str__(self) -> str:
        return f"Entry {self.entry_id} from {self.journal_id}"

    # analyse data and return sentiment score range from -1 to 1 by
    ↪  calling google cloud api
    def analyse_sentiment(self) -> float:
        client = language_v2.LanguageServiceClient()
        document = language_v2.Document(
            content=self.content,
            ↪  type_=language_v2.Document.Type.PLAIN_TEXT
        )
        response = client.analyze_sentiment(request={"document":
        ↪  document})
```

```
121         return response.document_sentiment.score
```

/backend/Entries/urls.py

```
1  from . import views
2  from django.urls import re_path
3
4  #
5  urlpatterns = [
6      re_path("sample/", views.sampleEntry, name="sample entry"),
7      re_path("testUser/", views.test_user, name="test user"),
8      re_path("getEntries/", views.get_all_entries, name="get all
       ↪  entries"),
9      re_path("createEntry/", views.create_entry, name="create entry"),
10     re_path(
11         "getScheduledUsers/",
12         views.get_daily_scheduled_email_users,
13         name="get scheduled email users",
14     ),
15     re_path("deleteEntr`y/", views.delete_entry, name="delete entry"),
16     re_path(
17         "getJournalStatistics/",
18         views.get_journal_statistics,
19         name="get entry statistics",
20     ),
21 ]
```

/backend/Entries/views.py

```
1  # Create your views here.
2  from urllib import response
3  from rest_framework.response import Response
```

```
4  from rest_framework.decorators import api_view
5  from rest_framework import status
6  from rest_framework.authtoken.models import Token
7
8  from rest_framework.permissions import IsAuthenticated
9  from rest_framework.decorators import authentication_classes,
   ↪  permission_classes
10 from rest_framework.authentication import TokenAuthentication,
   ↪  SessionAuthentication
11 from django.shortcuts import get_object_or_404
12
13 from Users.models import MyUser
14 from .models import Entry, Journal
15
16
17 @api_view(["get"])
18 def sampleEntry(request):
19     entry = [
20         {
21             "user_id": 1,
22             "entry_id": 1,
23             "creation_date": "2021-09-30",
24             "title": "Title One",
25             "content": "lorem ipsum dolor sit amet consectetur
                ↪  adipisicing elit. Quisquam, voluptatum.",
26         },
27         {
28             "user_id": 1,
29             "entry_id": 2,
30             "creation_date": "2021-09-30",
31             "title": "Title Two",
32             "content": "lorem ipsum dolor sit amet consectetur
                ↪  adipisicing elit. Quisquam, voluptatum.",
33         },
34         {
35             "user_id": 1,
36             "entry_id": 3,
37             "creation_date": "2021-09-30",
38             "title": "Title Three",
39             "content": "lorem ipsum dolor sit amet consectetur
                ↪  adipisicing elit. Quisquam, voluptatum.",
40         },
41     ]
42     return Response(entry, status=status.HTTP_200_OK)
43
```

```python

@api_view(["get"])
@authentication_classes([TokenAuthentication, SessionAuthentication])
@permission_classes([IsAuthenticated])
def test_user(request):
    user = request.user
    return Response(
        {"message": f"Hello {user.user_id}! You are logged in"},
        status=status.HTTP_200_OK,
    )


@api_view(["get"])
@authentication_classes([TokenAuthentication, SessionAuthentication])
@permission_classes([IsAuthenticated])
def get_all_entries(request):
    user = request.user
    journal = Journal.objects.get(user_id=user.user_id)
    entries = journal.get_all_entries()
    response = []
    for entry in entries:
        response.append(
            {
                "user_id": user.user_id,
                "entry_id": entry.entry_id,
                "creation_date": entry.creation_date,
                "title": entry.title,
                "content": entry.content,
            }
        )
    if len(response) == 0:
        return Response(
            {"error": "No entries found for this user."},
            status=status.HTTP_404_NOT_FOUND,
        )
    return Response(response, status=status.HTTP_200_OK)


@api_view(["post"])
@authentication_classes([TokenAuthentication, SessionAuthentication])
@permission_classes([IsAuthenticated])
def create_entry(request) -> Response:
    user = request.user
    if not Journal.objects.filter(user_id=user).exists():
        return Response(
```

```
            {"error": "Journal does not exist for this user."},
            status=status.HTTP_404_NOT_FOUND,
        )

    journal = Journal.objects.get(user_id=user)

    entry = Entry(
        journal_id=journal,
        title=request.data["title"],
        content=request.data["content"],
    )
    entry.save()
    return Response(
        {"message": "Entry created successfully"},
        ↪  status=status.HTTP_201_CREATED
    )


@api_view(["get"])
@authentication_classes([TokenAuthentication, SessionAuthentication])
@permission_classes([IsAuthenticated])
def delete_entry(request):
    user = request.user
    if not Journal.objects.filter(user_id=user).exists():
        return {"error": "can't find user's journal"}

    journal = Journal.objects.get(user_id=user)

    entry_id = request.data["entry_id"]

    if not Entry.objects.filter(journal_id=journal,
    ↪  entry_id=entry_id).exists():
        return Response(
            {"error": "Entry does not exist for this user."},
            status=status.HTTP_404_NOT_FOUND,
        )

    entry = Entry.objects.get(journal_id=journal, entry_id=entry_id)
    entry.delete()
    return Response(
        {"message": "Entry deleted successfully"},
        ↪  status=status.HTTP_200_OK
    )


```

```python
@api_view(["get"])
@authentication_classes([TokenAuthentication, SessionAuthentication])
@permission_classes([IsAuthenticated])
def get_journal_statistics(request):
    try:
        user = request.user
        if not Journal.objects.filter(user_id=user).exists():
            return Response(
                {"error": "Journal does not exist for this user."},
                status=status.HTTP_404_NOT_FOUND,
            )

        journal = Journal.objects.get(user_id=user)
        most_active_month_int = journal.get_most_active_month()
        average_entries_per_week = journal.get_average_entries_per_week()
        total_word_count = journal.get_total_word_count()

        match most_active_month_int:
            case 1:
                most_active_month = "January"
            case 2:
                most_active_month = "February"
            case 3:
                most_active_month = "March"
            case 4:
                most_active_month = "April"
            case 5:
                most_active_month = "May"
            case 6:
                most_active_month = "June"
            case 7:
                most_active_month = "July"
            case 8:
                most_active_month = "August"
            case 9:
                most_active_month = "September"
            case 10:
                most_active_month = "October"
            case 11:
                most_active_month = "November"
            case 12:
                most_active_month = "December"
            case _:
                most_active_month = "No entries found"

```

```
        return Response(
            {
                "most_active_month": most_active_month,
                "average_entries_per_week": average_entries_per_week,
                "total_word_count": total_word_count,
            },
            status=status.HTTP_200_OK,
        )
    except Exception as e:
        return Response(
            {"error": str(e)},
            status=status.HTTP_500_INTERNAL_SERVER_ERROR,
        )


@api_view(["get"])
def get_daily_scheduled_email_users(request):
    """
    This method to get the list of users who have email_prompt set to
    ↪  True. I wrote this method to test wether I could get the list of
    ↪  users who have email_prompt set to True through calling the API.
    ↪  This works as expected so now I use this code in my email app.
    """

    users = MyUser.objects.all()
    user_list = [user.email for user in users if user.email_prompt is
    ↪  True]

    print(user_list)
    return Response(
        {
            "users": user_list,
        },
        status=status.HTTP_200_OK,
    )
```

/frontend/src/main.tsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.tsx";
import "./index.css";
import { UserProvider } from "./context/userContext.tsx";
// import { QueryClient, QueryClientProvider } from "react-query";
import { BrowserRouter } from "react-router-dom";

ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>
    <UserProvider>
      <BrowserRouter>
        <App />
      </BrowserRouter>
    </UserProvider>
  </React.StrictMode>
);
```

/frontend/src/App.tsx

```
import "./App.css";
import NavBar from "./components/layout/NavBar";
import { Route, Routes } from "react-router-dom";
import useUserContext from "./hooks/useUserContext";

import HomePage from "./pages/HomePage";
import LoginPage from "./pages/login/LoginPage";
import RegisterPage from "./pages/register/RegisterPage";
import ProfilePage from "./pages/ProfilePage";
import JournalEntriesPage from "./pages/entry/JournalEntriesPage";
import CreateEntryPage from "./pages/entry/CreateEntryPage";
import StatisticsPage from "./pages/statistics/StatisticsPage";

import Cookies from "universal-cookie";
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";

```

```
function App() {
  const { user } = useUserContext();

  // Check if the user is logged in
  const cookies = new Cookies();

  return (
    <div className="App">
      <div className="routes">
        <NavBar isAuthenticated={!user ? false : true} />
        <Routes>
          <Route path="/" element={<HomePage />} />
          <Route path="/login" element={<LoginPage />} />
          <Route path="/register" element={<RegisterPage />} />
          <Route path="/profile" element={<ProfilePage />} />
          <Route path="/entries" element={<JournalEntriesPage />} />
          <Route path="/create-entry" element={<CreateEntryPage />} />
          <Route path="/statistics" element={<StatisticsPage />} />

          <Route
            path="/profile"
            element={user ? <ProfilePage /> : <RedirectToLogin />}
          />
        </Routes>
      </div>
    </div>
  );
}

// Redirect to login page if user is not logged in
function RedirectToLogin() {
  const navigate = useNavigate();

  useEffect(() => {
    setTimeout(() => {
      navigate("/login");
    }, 2000); // Redirect after 2 seconds
  }, [navigate]);

  return <div>You are not logged in. Redirecting to login page...</div>;
}

export default App;
```

/frontend/src/componets/layout/NavBar.tsx

```
import { Link } from "react-router-dom";
import "./NavBar.css";
import useAuth from "../../hooks/useAuth";

export default function NavBar({
  isAuthenticated,
}: {
  isAuthenticated: boolean;
}) {
  const { logout } = useAuth();
  return (
    <nav>
      <ul>
        {/* always display Home irrespectable of wether signed in*/}
        <li>
          <Link to="/">Home</Link>
        </li>

        {/* conditionally display login/register or profile/logout */}
        {isAuthenticated ? (
          <>
            <li>
              <Link to="/profile">Profile</Link>
            </li>
            <li>
              <Link to="/entries">Entries</Link>
            </li>
            <li>
              <Link to="/create-entry">Create</Link>
            </li>
            <li>
              <button onClick={logout}>Logout</button>
            </li>
          </>
        ) : (
          <>
            <li>
              <Link to="/login">Login</Link>
            </li>
            <li>
              <Link to="/register">Register</Link>{" "}
```

```
            </li>
          </>
        )}
      </ul>
    </nav>
  );
}
```

/frontend/src/context/userContext.tsx

```
import Cookies from "universal-cookie";
import { User } from "../hooks/useUser";
import React, { createContext, useState, ReactNode, useEffect } from
↪  "react";

export interface UserContextType {
  user: User | null;
  updateUser: (user: User | null) => void;
}

export const UserContext = createContext<UserContextType | undefined>(
  undefined
);

export const UserProvider: React.FC<{ children: ReactNode }> = ({
  children,
}) => {
  const cookies = new Cookies();
  const [user, setUser] = useState<User | null>(() => {
    // prevent my user being lost on refresh
    const savedUser = cookies.get("user");
    if (savedUser) {
      return savedUser;
    }
  });

  const updateUser = (newUser: User | null) => {
    setUser(newUser);
```

```
    console.log("update user for context: ", newUser, user);
  };

  useEffect(() => {
    if (user) console.log("user context: ", user);
  }, [user]);

  return (
    <UserContext.Provider value={{ user, updateUser }}>
      {children}
    </UserContext.Provider>
  );
};
```

/frontend/src/hooks/useAuth.ts

```
import { User } from "./useUser";
import useUser from "./useUser";

export default function useAuth() {
    const { getUser, addUser, removeUser } = useUser();

    const login = (user: User) => {
        addUser(user);
    }

    const logout = () => {
        removeUser();
    }

    const isAuthenticated = () => {
        return getUser() !== undefined;
    }

    return {
        login,
        logout,
        isAuthenticated
```

```
    }}
```

/frontend/src/hooks/useFetchData.tsx

```
import { useState, useEffect } from "react";
import axios from "axios";

const API_URL = "http://127.0.0.1:8000";

interface Post {
  user_id: number;
  entry_id: number;
  creation_date: string;
  title: string;
  body: string;
}

export function useGetPostsAPI({
  path,
  token,
}: {
  path: string;
  token?: string | null;
}): { isLoading: boolean; error: any; posts: Post[] } {
  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState();
  const [posts, setPosts] = useState<Post[]>([]);

  useEffect(() => {
    const fetchPosts = async () => {
      setIsLoading(true);

      const headers: { [key: string]: string } = {
        "Content-Type": "application/json",
      };
      if (token) {
        headers["Authorization"] = `Token ${token}`;
      }
```

```
35
36        try {
37          const response = await axios.get(`${API_URL}${path}`, { headers
              ↪  });
38          const posts = response.data as Post[];
39          setPosts(posts);
40          console.log(posts);
41        } catch (error: any) {
42          if (axios.isCancel(error)) {
43            return;
44          }
45
46          setError(error);
47        } finally {
48          setIsLoading(false);
49        }
50      };
51      fetchPosts();
52    }, [path]);
53
54    return { isLoading, error, posts };
55  }
```

/frontend/src/hooks/useUser.ts

```
1   import Cookies from "universal-cookie";
2   import useUserContext from "./useUserContext";
3
4   export interface User {
5       user_id: number;
6       email: string;
7       first_name: string;
8       last_name: string;
9       authToken?: string;
10      email_prompt: boolean;
11    }
12  export default function useUser() {
13      const { updateUser } = useUserContext();
```

```
14
15      const cookies = new Cookies();
16
17      const addUser = (user: User) => {
18          cookies.set("user", user, { path: "/",
19          secure: true,
20          sameSite: "strict"
21
22      });
23          updateUser(user);
24          console.log("User added");
25      }
26
27      const removeUser = () => {
28          cookies.remove("user");
29          updateUser(null);
30          console.log("User removed");
31      }
32      const getUser = (): User | undefined  => {
33          return cookies.get("user");
34      }
35
36      return {
37          addUser,
38          removeUser,
39          getUser
40      }
41  }
```

/frontend/src/hooks/useUserContext.ts

```
1   import { useContext} from "react";
2   import { UserContext, UserContextType  } from "../context/userContext";
3
4   export default function useUserContext() {
5       const context = useContext(UserContext) as UserContextType;
6
7       if (context === undefined) {
```

```
        throw new Error("useUser must be used within a UserProvider");
    }

    return context;

}
```

/frontend/src/pages/HomePage.tsx

```
import { useNavigate } from "react-router-dom";

function HomePage() {
  const navigate = useNavigate();
  const goToCreateEntry = () => {
    console.log("Go to create entry");
    navigate("/create-entry");
  };
  const goToViewEntries = () => {
    console.log("Go to view entries");
    navigate("/entries");
  };

  return (
    <div className="home-page">
      <h1>Home</h1>
      <p>Welcome to the home page</p>
      <button onClick={goToCreateEntry}>Create Entry</button>
      <button onClick={goToViewEntries}>View Entries</button>
    </div>
  );
}

export default HomePage;
```

/frontend/src/pages/ProfilePage.tsx

```
import { useNavigate } from "react-router-dom";
import { useEffect, useState } from "react";
import { User } from "../hooks/useUser";
import useUser from "../hooks/useUser";
// import useApi from "../api/useApi";

// interface requestFeedback {
//   message: string;
// }

export default function ProfilePage() {
  const navigate = useNavigate();
  const { getUser } = useUser();
  // const [isChecked, setIsChecked] = useState<boolean | null>(null);
  const [myUser, setMyUser] = useState<User | null>(null);
  // const { isLoading, error, data, fetchData } =
  ↪  useApi<requestFeedback>({
  //   url: "/user/update_email_prompt/",
  //   method: "POST",
  //   data: {
  //     email_prompt: isChecked,
  //   },
  //   headers: {
  //     Authorization: `Token ${myUser?.authToken}`,
  //   },
  // });

  // function updateEmailJournal() {
  //   console.log("Updating email journal");
  //   const myToken = myUser?.authToken;
  //   fetchData(); // This will trigger the API call
  // }

  useEffect(() => {
    const user = getUser();
    if (user) {
      setMyUser(user);
    } else {
      navigate("/login");
    }
  }, []);
```

```
41
42  if (!myUser) {
43    return <div>user not found</div>;
44  } else {
45    return (
46      <div>
47        <h1>Profile</h1>
48        <div>
49          <div>
50            <div>
51              <label htmlFor="id:">User ID:</label>
52              <input
53                type="text"
54                id="id"
55                name="id"
56                value={myUser.user_id}
57                readOnly
58              />
59            </div>
60            <div>
61              <label htmlFor="email">Email:</label>
62              <input
63                type="email"
64                id="email"
65                name="email"
66                value={myUser.email}
67                readOnly
68              />
69            </div>
70            <label htmlFor="name">Name:</label>
71            <input
72              type="text"
73              id="name"
74              name="name"
75              value={myUser.first_name + " " + myUser.last_name}
76              readOnly
77            />
78          </div>
79          <div>
80            <label htmlFor="name">Auth token:</label>
81            <input
82              type="text"
83              id="name"
84              name="name"
85              value={myUser.authToken}
```

```
              readOnly
            />

            <div>
              <button onClick={() => navigate("/statistics")}>
                View Statistics
              </button>
            </div>

            {/* <div className="entries"></div>
            <div className="emailJournal">
              <label htmlFor="emailJournal">
                Would you like daily Journal Email feature:
              </label>
              <div className="emailJournalCheckbox">
                <input
                  type="checkbox"
                  id="emailJournal"
                  name="emailJournal"
                  checked={myUser.email_prompt}
                  onChange={(event) => {
                    setIsChecked(event.target.checked);
                  }}
                />
                {isChecked ? "yes" : "no"}
                <input
                  disabled={isLoading}
                  type="button"
                  value="save"
                  onClick={updateEmailJournal}
                />
              </div>
              {error && <div>Error: {error.message}</div>}
              {isLoading && <div>Loading...</div>}
            </div> */}
          </div>
        </div>
      </div>
    );
  }
}
```

/frontend/src/pages/entry/CreateEntryPage.tsx

```tsx
import { useState } from "react";
import useApi from "../../api/useApi";
import useUserContext from "../../hooks/useUserContext";
import { useNavigate } from "react-router-dom";

interface requestFeedback {
  message: string;
}

export default function CreateEntryPage() {
  const { user } = useUserContext();
  const myToken = user?.authToken;

  const [title, setTitle] = useState("");
  const [content, setContent] = useState("");

  const navigate = useNavigate();

  const { isLoading, error, data, fetchData } = useApi<requestFeedback>({
    url: "/entry/createEntry/",
    method: "POST",
    data: {
      title,
      content,
    },
    headers: {
      Authorization: `Token ${myToken}`,
    },
  });

  if (user === null || user === undefined) {
    return (
      <>
        <div>You are not logged in please either login or register</div>
        <button onClick={() => navigate("/login")}>Login</button>
        <button onClick={() => navigate("/register")}>Register</button>
      </>
    );
  }

  const onCreateEntry = () => {
```

```
    fetchData();
    console.log("Creating entry...");
    if (data) {
      navigate("/entries");
    }
  };

  return (
    <div>
      <h1>Create Journal Entry</h1>
      <div>
        <label htmlFor="title">Title:</label>
        <input
          type="text"
          id="title"
          name="title"
          value={title}
          onChange={(event) => setTitle(event.target.value)}
        />
      </div>
      <div>
        <label htmlFor="content">Content:</label>
        <textarea
          id="content"
          name="content"
          value={content}
          onChange={(event) => setContent(event.target.value)}
        />
      </div>
      {error && <div>Error: {error.message}</div>}
      {isLoading && <div>Loading...</div>}
      {data && <div>{data.message}</div>}
      <input type="submit" value="Create Entry" onClick={onCreateEntry}
      ↪ />
    </div>
  );
}
```

/frontend/src/pages/entry/JournalEntiesyPage.tsx

```
import useApi from "../../api/useApi";
import useUserContext from "../../hooks/useUserContext";
import { useEffect, useState } from "react";
import { sortEntriesByDate } from "./sortEntries";
import { useNavigate } from "react-router-dom";

export interface Entry {
  entry_id: number;
  title: string;
  content: string;
  creation_date: string;
}

export default function JournalEntriesPage() {
  const { user } = useUserContext();
  const myToken = user?.authToken;

  const navigate = useNavigate();

  const [sortOrder, setSortOrder] = useState("desc");
  const [entries, setEntries] = useState<Entry[] | null>(
    localStorage.getItem("journalEntries")
      ? JSON.parse(localStorage.getItem("journalEntries") as string)
      : null
  );

  // Add state for current page and items per page
  const [currentPage, setCurrentPage] = useState(1);
  const itemsPerPage = 5;

  // Calculate the index of the first and last items on the current page
  const indexOfLastItem = currentPage * itemsPerPage;
  const indexOfFirstItem = indexOfLastItem - itemsPerPage;

  // Slice the array of entries to get only the items for the current
  ↪  page, empty array if no entries
  const currentItems = sortEntriesByDate(entries, sortOrder).slice(
    indexOfFirstItem,
    indexOfLastItem
  );

```

```
41    // Calculate the total number of pages
42    const totalPages = entries ? Math.ceil(entries.length / itemsPerPage) :
      ↪  1;
43
44    // Add handlers for the pagination buttons
45    const goToNextPage = () => {
46      setCurrentPage((page) => Math.min(page + 1, totalPages));
47    };
48
49    const goToPreviousPage = () => {
50      setCurrentPage((page) => Math.max(page - 1, 1));
51    };
52
53    // defining how to fetch data from my API and the format to put it in
54    const { isLoading, error, data, fetchData } = useApi<Entry[]>({
55      url: "/entry/getEntries/",
56      method: "get",
57      headers: {
58        Authorization: `Token ${myToken}`,
59      },
60    });
61    console.log("user", user);
62
63    if (user === null || user === undefined) {
64      return (
65        <>
66          <div>You are not logged in please either login or register</div>
67          <button onClick={() => navigate("/login")}>Login</button>
68          <button onClick={() => navigate("/register")}>Register</button>
69        </>
70      );
71    }
72
73    async function updateEntries() {
74      await fetchData();
75      localStorage.setItem("journalEntries", JSON.stringify(data));
76      entries && setEntries(data);
77    }
78
79    // check local storage to see if we already have journal data:
80    useEffect(() => {
81      if (data !== null) {
82        setEntries(data);
83        localStorage.setItem("journalEntries", JSON.stringify(data));
84      }
```

```
85    }, [data]);
86
87    // if we have journal data in local storage, use that, otherwise use
      ↪  the data from the API
88    if (entries === null) {
89      return (
90        <>
91          <div>No entries found</div>
92          <button onClick={updateEntries}>Update Entries</button>
93        </>
94      );
95    } else {
96      return (
97        <>
98          <h1>Journal Entries</h1>
99          <div>
100           <button onClick={updateEntries}>Update Entries</button>
101
102           <h2>Sort by Date</h2>
103           <button onClick={() => setSortOrder("desc")}>Desending</button>
104           <button onClick={() => setSortOrder("asc")}>Ascending</button>
105         </div>
106         <ul>
107           {sortEntriesByDate(currentItems, sortOrder).map((post) => (
108             <li key={post.entry_id}>
109               <h2>{post.title}</h2>
110               <p>
111                 date: {post.creation_date}
112                 <br />
113                 {post.content}
114               </p>
115             </li>
116           ))}
117         </ul>
118         <div>
119           <p>
120             Page {currentPage} of {totalPages}
121           </p>
122           <button onClick={goToPreviousPage}>Previous</button>
123           <button onClick={goToNextPage}>Next</button>
124         </div>
125         {error && <div>Error: {error.message}</div>}
126         {isLoading && <div>Loading...</div>}
127       </>
128     );
```

```
  }
}
```

/frontend/src/pages/entry/sortEntries.ts

```
import { Entry} from "./JournalEntriesPage";


export function sortEntriesByDate(entries: Entry[] | null, sortOrder:
↪  string): Entry[] {
  // defensive programming
  if (!entries) {
    return Array<Entry>();
  }
  // zipping the entries with their creation date
  const zippedEntries = entries.map((entry) => {
    return [entry, new Date(entry.creation_date).getTime()] as [Entry,
    ↪  number];
  })

  timSort(zippedEntries);

  // returns the sorted entries in unzipped format
  if (sortOrder === "desc") {
    return zippedEntries.map((zippedEntry) => zippedEntry[0]).reverse();
  }
  return zippedEntries.map((zippedEntry) => zippedEntry[0]);
}


function insertionSort<Key>(
  arr: Array<[Entry, Key]>,
  startIndex: number,
  endIndex: number,
) {

  // iterate the segment of the array from startIndex to endIndex
  for (let i = startIndex + 1; i < endIndex; i++) {
```

```
32      // increment the currently searching index i with each pass, and use
        ↪  j as a temprary index to compare the current element with all
        ↪  previous elements
33      let j = i;
34
35      while (j > startIndex && arr[j - 1][1] > arr[j][1]) {
36
37        // compare the current element with every single previous element
          ↪  and keep swapping until it is in the right position
38        swap(arr, j, j - 1);
39        j--;
40      }
41    }
42  }
43
44  function swap<T>(arr: Array<[Entry, T]>, i: number, j: number) {
45    const temp = arr[i];
46    arr[i] = arr[j];
47    arr[j] = temp;
48  }
49
50
51  function merge<Key>(arr: Array<[Entry, Key]>, left: number, mid: number,
    ↪  right: number) {
52    // the two different segments sorted by insertion sort need to be
      ↪  merged, the left segment is from left to mid, and the right segment
      ↪  is from mid to right
53    const leftArr = arr.slice(left, mid);
54    const rightArr = arr.slice(mid, right);
55
56
57    // i is the index of the left segment, j is the index of the right
      ↪  segment, and k is the index of the merged segment
58    let i = 0;
59    let j = 0;
60    let k = left;
61
62    // going through all the elements of the left and right segments and
      ↪  compare them, and merge them in the right order
63    while (i < leftArr.length && j < rightArr.length) {
64
65      // basically means if the left element is smaller than the right
        ↪  element, then put the left element in the merged segment,
        ↪  otherwise put the right element in the merged segment in that
        ↪  sequence
```

```
    if (leftArr[i][1] <= rightArr[j][1]) {
      arr[k] = leftArr[i];
      i++;
    } else {
      arr[k] = rightArr[j];
      j++;
    }
    k++;
  }

  // if there are any remaining elements in the left or right segment, then put them in the merged segment
  while (i < leftArr.length) {
    arr[k] = leftArr[i];
    i++;
    k++;
  }

  while (j < rightArr.length) {
    arr[k] = rightArr[j];
    j++;
    k++;
  }
}

function timSort<Key>(zippedEntries: Array<[Entry, Key]>){
  // run means the size of the segments that will be sorted by insertion sort
  const RUN = 16;
  const n = zippedEntries.length;

  // use insertion sort on all the individual runs with the size of RUN
  for (let i = 0; i < n; i += RUN) {
    // math.min is used so that the sorting does not go out of bounds
    insertionSort(zippedEntries, i, Math.min(i + RUN, n));
  }


  // merge all the the sorted segments
  for (let size = RUN; size < n; size = 2 * size) {
    for (let left = 0; left < n; left += 2 * size) {
      const mid = left + size;
      const right = Math.min(left + 2 * size, n);
      merge(zippedEntries, left, mid, right);
    }
```

```
  }
}
```

/frontend/src/pages/login/LoginPage.tsx

```
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import useApi from "../../api/useApi";
import { User } from "../../hooks/useUser";
import useAuth from "../../hooks/useAuth";

export default function LoginPage(): JSX.Element {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const { isAuthenticated, login } = useAuth();
  const navigate = useNavigate();

  // https://dmitripavlutin.com/react-useeffect-explanation/
  useEffect(() => {
    if (isAuthenticated() === true) {
      navigate("/profile");
    }
  }, [isAuthenticated]);

  const { isLoading, error, data, fetchData } = useApi<User>({
    url: "/user/login/",
    method: "post",
    data: {
      email,
      password,
    },
  });

  useEffect(() => {
    if (data) {
      login(data);
      navigate("/profile");
    }
```

```
  }, [data]);

  const onLoginFunc = () => {
    console.log("Logging in...");
    fetchData();
  };

  return (
    <div>
      <input
        placeholder="email"
        value={email}
        onChange={(event) => setEmail(event.target.value)}
      />
      <input
        type="password"
        placeholder="password"
        value={password}
        onChange={(event) => setPassword(event.target.value)}
      />
      {isLoading && <div>Loading...</div>}
      {error && <div>{error.message}</div>}
      {error && <div>{JSON.stringify(error.response?.data)}</div>}
      {data && <div>Logged in!</div>}
      {data && <div>{JSON.stringify(data)}</div>}

      <button onClick={onLoginFunc}>Login</button>
    </div>
  );
}
```

/frontend/src/pages/register/RegisterPage.tsx

```
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import useAuth from "../../hooks/useAuth";
import useApi from "../../api/useApi";
import { User } from "../../hooks/useUser";
```

```
6
7  interface registerRequestData {
8    email: string;
9    password: string;
10   first_name: string;
11   last_name: string;
12 }
13
14 export default function RegisterPage(): JSX.Element {
15   // my states
16   const [registerRequestData, setregisterRequestData] =
17     useState<registerRequestData>({} as registerRequestData);
18   const navigate = useNavigate();
19   const { login } = useAuth();
20
21   const { isLoading, error, data, fetchData } = useApi<User>({
22     url: "/user/register/",
23     method: "post",
24     data: {
25       email: registerRequestData.email,
26       password: registerRequestData.password,
27       first_name: registerRequestData.first_name,
28       last_name: registerRequestData.last_name,
29     },
30   });
31
32   useEffect(() => {
33     if (data) {
34       // redirect to profile page
35       login(data);
36       navigate("/profile");
37     }
38   }, [data]);
39   const onRegisterFunc = () => {
40     console.log("Registering...");
41     fetchData(); // triggers myApiCall
42   };
43   return (
44     <div>
45       <input
46         placeholder="email"
47         value={registerRequestData.email}
48         onChange={(e) =>
49           setregisterRequestData({
50             ...registerRequestData,
```

```
            email: e.target.value,
          })
        }
      />
      <input
        type="password"
        placeholder="password"
        value={registerRequestData.password}
        onChange={(e) =>
          setregisterRequestData({
            ...registerRequestData,
            password: e.target.value,
          })
        }
      />
      <input
        placeholder="first name"
        value={registerRequestData.first_name}
        onChange={(e) =>
          setregisterRequestData({
            ...registerRequestData,
            first_name: e.target.value,
          })
        }
      />
      <input
        placeholder="last name"
        value={registerRequestData.last_name}
        onChange={(e) =>
          setregisterRequestData({
            ...registerRequestData,
            last_name: e.target.value,
          })
        }
      />
      {isLoading && <div>Loading...</div>}
      {error && <div>{error.message}</div>}
      {error && <div>{JSON.stringify(error.response?.data)}</div>}
      {data && (
        <div>
          Registered successfully
          <div>Welcome {data.first_name}</div>
        </div>
      )}

```

```
      <br />

      <input
        disabled={isLoading}
        type="button"
        value="Register"
        onClick={onRegisterFunc}
      />
    </div>
  );
}
```

SourceCode/frontend/src/pages/statistics/StatisticsPage.tsx

```
import { useEffect } from "react";
import useApi from "../../api/useApi";
import useUserContext from "../../hooks/useUserContext";

export interface Statistics {
  most_active_month: string;
  average_entries_per_week: number;
  total_word_count: number;
}

export default function StatisticsPage() {
  const { user } = useUserContext();
  const myToken = user?.authToken;

  const { isLoading, error, data, fetchData } = useApi<Statistics>({
    url: "/entry/getJournalStatistics/",
    method: "GET",
    headers: {
      Authorization: `Token ${myToken}`,
    },
  });

  useEffect(() => {
    fetchData();
```

```
25    }, []);
26
27    if (isLoading) {
28      return <div>Loading...</div>;
29    }
30
31    if (error) {
32      return (
33        <>
34          <div>Error: {error.message}</div>
35          <div>{JSON.stringify(error.response?.data)}</div>
36        </>
37      );
38    }
39
40    if (!data) {
41      return <div>Data not found</div>;
42    }
43
44    return (
45      <div>
46        <h1>Statistics</h1>
47        <div>
48          <div>
49            <label htmlFor="most_active_month">Most active month:</label>
50            <input
51              type="text"
52              id="most_active_month"
53              name="most_active_month"
54              value={data.most_active_month}
55              readOnly
56            />
57          </div>
58          <div>
59            <label htmlFor="average_entries_per_week">
60              Average entries per week:
61            </label>
62            <input
63              type="text"
64              id="average_entries_per_week"
65              name="average_entries_per_week"
66              value={data.average_entries_per_week}
67              readOnly
68            />
69          </div>
```

```
        <div>
          <label htmlFor="total_word_count">Total word count:</label>
          <input
            type="text"
            id="total_word_count"
            name="total_word_count"
            value={data.total_word_count}
            readOnly
          />
        </div>
      </div>
    </div>
  );
}
```

Bibliography

[1] Iso week date. https://en.wikipedia.org/wiki/ISO_week_date.

[2] Django rest framework, 2024.

[3] Beau Carnes. Uml diagram course – how to design databases and systems. https://www.freecodecamp.org/news/uml-diagrams-full-course/, April 2021. Includes embedded YouTube video content.

[4] Alexander S. Gillis. What is a thick client (fat client)? - definition from whatis.com. https://www.techtarget.com/whatis/definition/fat-client-thick-client, 2020. Accessed: 2024-03-17.

[5] Nazish Imran, Irum Aamer, Muhammad Imran Sharif, Zubair Hassan Bodla, and Sadiq Naveed. Psychological burden of quarantine in children and adolescents: A rapid systematic review and proposed solutions. *Pakistan Journal of Medical Sciences*, 36(5), 2020.

[6] Sarah Laoyan. What is agile methodology? (a beginner's guide). https://asana.com/resources/agile-methodology, February 2024.

[7] OCR. A Level Further Mathematics A Formulae Booklet. https://www.ocr.org.uk/Images/308765-a-level-further-mathematics-a-formulae-booklet.pdf, 2019.