Computing NEA - Journal App

Xichao Wang

March 19, 2024

Contents

1 Abstract **3**

2 Analysis **4**
- 2.1 Problem Area 4
  - 2.1.1 Benefits of Journalling 5
- 2.2 Client / End User 5
  - 2.2.1 Survey Result 6
- 2.3 Research Methodology 8
- 2.4 Features of Proposed Solution 8
- 2.5 Requirements Specification 9
- 2.6 Critical Path 11

3 Design **12**
- 3.1 Introduction 12
- 3.2 Frontend 12
- 3.3 Backend 14
  - 3.3.1 ASP.NET: 14
  - 3.3.2 Next.js and Express.js: 14
  - 3.3.3 Flask and FastAPI: 15
  - 3.3.4 Django 15
- 3.4 Hierarchy Charts 15
  - 3.4.1 Django Server Design: 16
  - 3.4.2 React Frontend 16
  - 3.4.3 Flowchart 19
- 3.5 Data Structures / Data Modelling 20
  - 3.5.1 Entity Relationship Modelling 20
  - 3.5.2 Abstract Data Structures 22
  - 3.5.3 External Data Sources 22
  - 3.5.4 OOP Model 22

3.6 File / Table Structures 23
3.7 Algorithms 24
3.8 User Interface 25
3.8.1 Wireframe Design 25
3.9 RESTful API Endpoints 29
3.10 Hardware & Software Requirements 29
3.11 User Authentication 29
3.11.1 Session Based Authentication 29

4 Technical Implementation 30
4.1 Key Code Segments 30
4.2 Data Structures 30
4.3 Algorithms 30
4.4 Modularity 31
4.4.1 Virtual Environment 31
4.5 Defensive Programming / Robustness 31
4.5.1 Server Based Authentication 31
4.6 Patterns 31
4.6.1 Single Responsibility Principal 31
4.7 Credential Management 31

5 Testing 32
5.1 Test Strategy 32
5.1.1 Rest Client 32
5.1.2 Django Test Models 32
5.2 Testing Video 32
5.3 System Tests 32

6 Evaluation 33
6.1 Requirements Specification Evaluation 33
6.2 Independent End-User Feedback 33
6.3 Improvements 33

7 Appendix 34
7.1 Code Listing 34
7.1.1 Appendix A – Code Listing 34

Chapter 1

Abstract

This document presents the A Level NEA Project, providing an overview of the problem area, the proposed solution's features, requirements specification, critical path, design details, technical implementation, testing strategy, evaluation, and an appendix containing code listings. [**einstein**]

This document presents my A level NEA project, which includes the following section with the aim to create a full picture

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 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, [**Imran˙Aamer˙Sharif˙Bodla˙Naveed˙2020**] it has left a void for individuals needing to relieve their anxieties and reduce the overall stress incorporated into their daily lives. Journalling 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 Journalling

Here are some benefits that I thought of:

- 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

Journalling is an excellent habit anyone could incorporate into their daily lives. However, my primary target end-user for my app would be teenagers like myself who need a safe and easily accessible place to document their lives and jot down their thoughts, emotions and goals.

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 journalling 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 which prevents me from cultivating the habit of journalling.

2.2.1 Survey Result

I have created a survey for friends who are my age to fill out

Interview

I had an interview with a teacher at my school who represents the someone who's slightly older than me and represent the wider user base of the app. I have picked out some responses which is particularly relevant for 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 user but also encrypting the data that is stored in 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 journalling] would be very useful for people on the go."

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

Q5. *Any additonal 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. Th 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 which introduced the concept of journalling, it highligthed the vast array of benefits that journalling can bring to an individual. Then, I looked through various online articles to understand the concept abit more.

A reason for the creation of the app was because I wanted to create a habit of journalling, 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.	Demonstrate through UI screenshots or video, and code snippets.
1.2	Implement a registration page enabling new users to create an account, take in input for email, password, last name, first name and interact/update backend. Provide feedback to the user after submission.	Demonstrate through UI screenshots or video, and code snippets.
2.1	Create a page for users to compose new journal entries. Input parameters are 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 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 example 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, 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 stored in the database, ensuring data stored are in the required format.	Code snippet and testing of the function that checks and validates the format of the provided data.
2.1	Develop RESTful API endpoints with functionalities implemented to handle user authentication (login and registration) and other core endpoints that drives 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	Apply best practices for securing sensitive data in the database and during data transmission.	
4.1	Design the backend architecture for scalability, considering future feature expansions and potential increases in user base.	
4.2	Document the API endpoints, data models, and any critical backend logic for maintenance and future development purposes.	

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. The Agile methodology would be suitable one I will be following for my project. This is because it is a flexible and iterative approach which allows repeated test of all the small modules and eventually building 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.

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 application tailored to these frameworks

3.2 Frontend

Frontend primarily refers to the interface 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, frontend is not only the UI, as it also includes the logic and functionality that drives 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 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 [**gillis˙2020˙fat**]. 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 really 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 have created, I had to separate my application for 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.

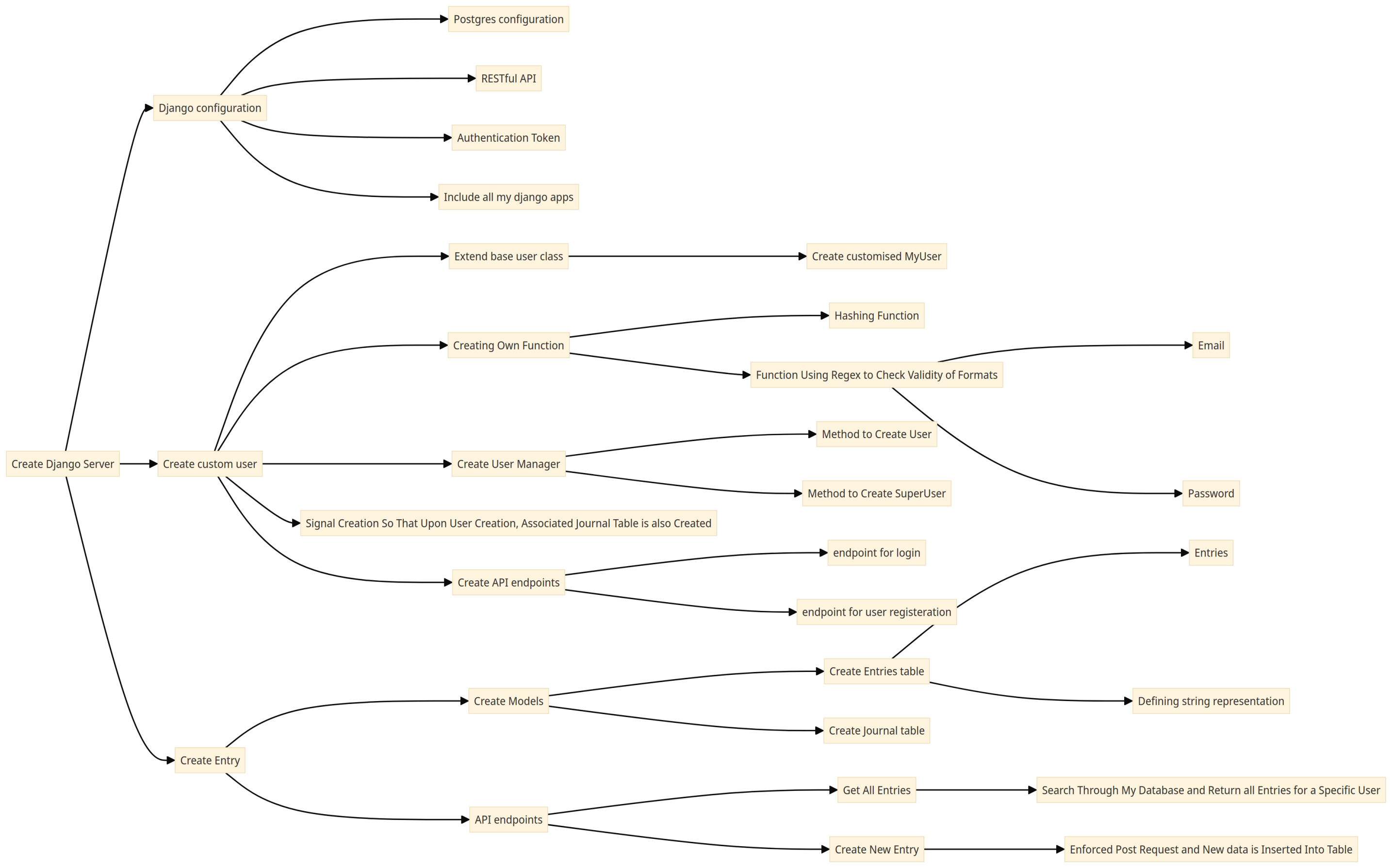


Figure 3.1: Backend Hierarchy Diagram

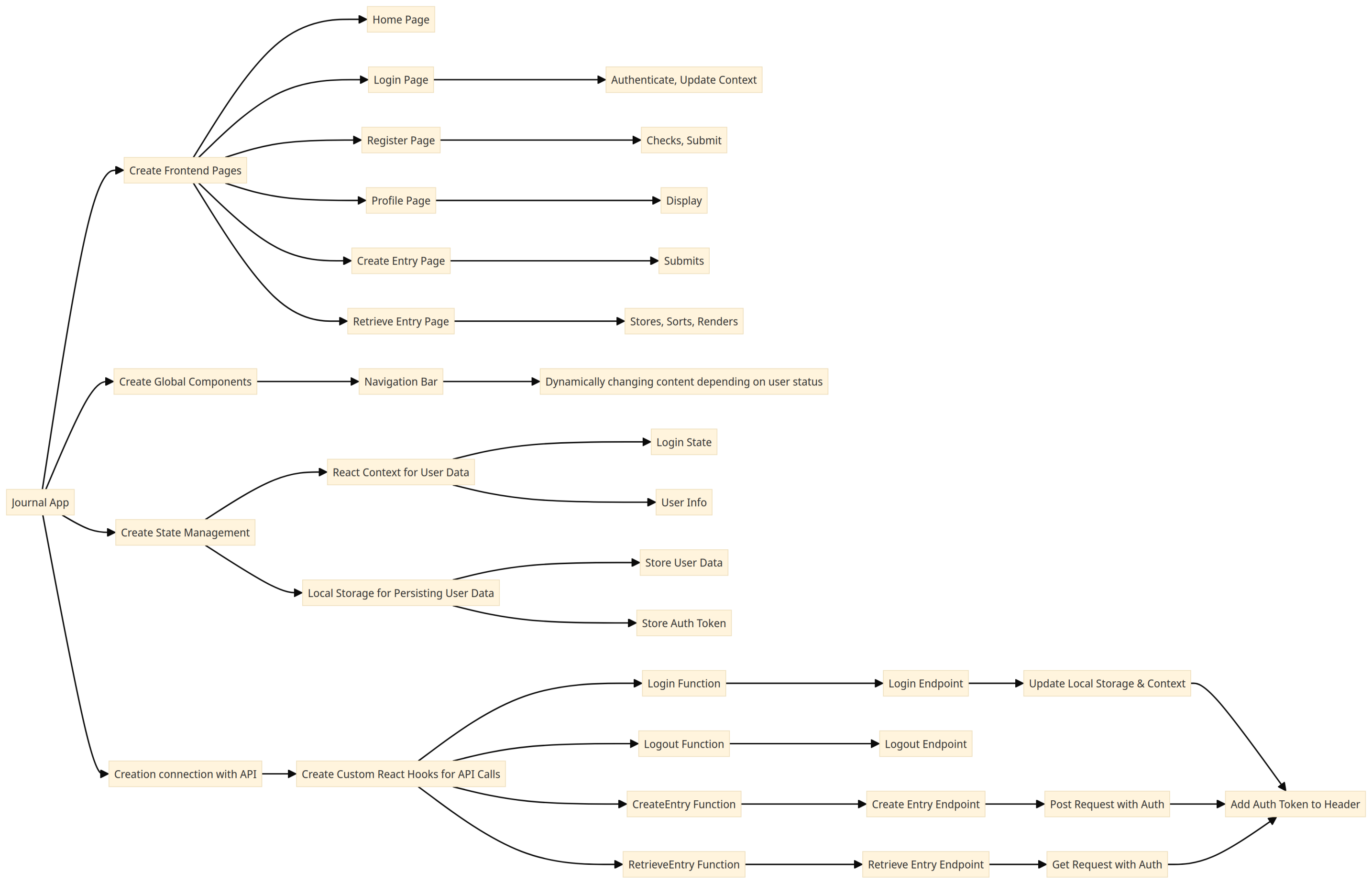


Figure 3.2: Frontend Hierarchy Diagram

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 front end also manipulates the storage system in the browser while the server handles the database.

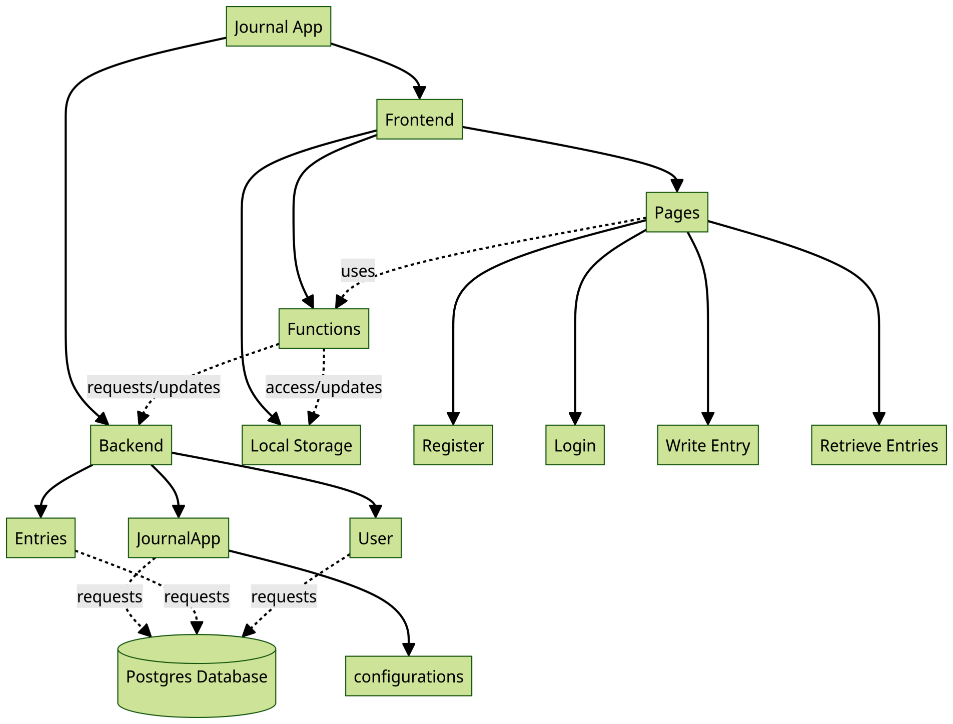


Figure 3.3: Flowchart of the System

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 of 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

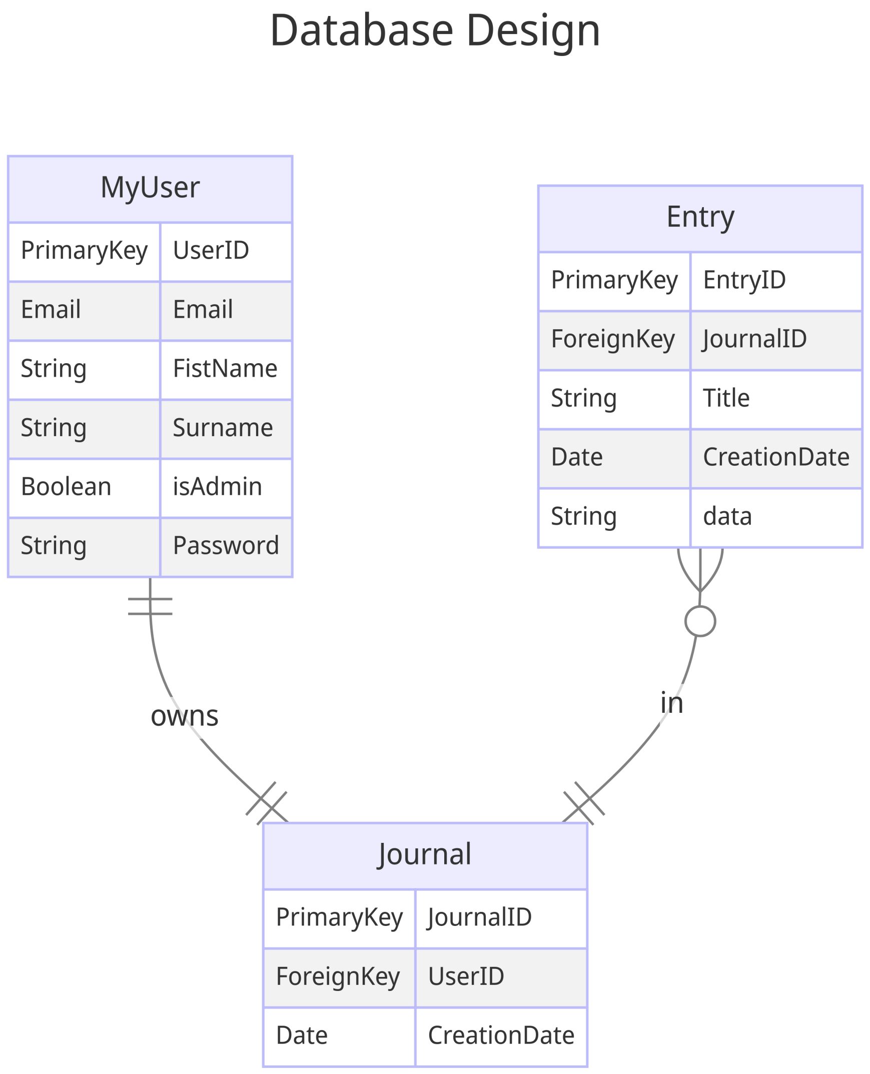


Figure 3.4: Entity Relationship Diagram

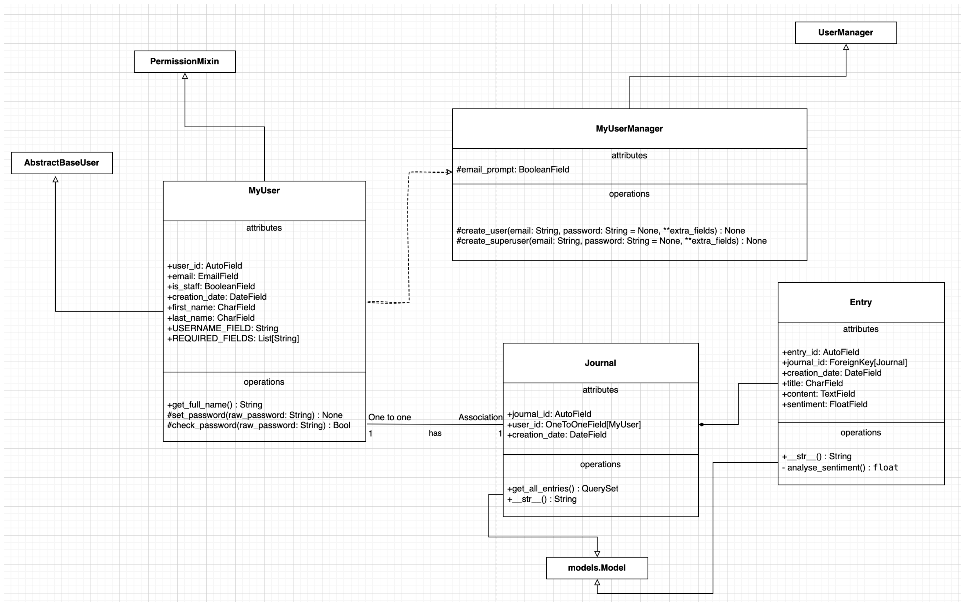
3.5.2 Abstract Data Structures

3.5.3 External Data Sources

Discuss any external data sources utilized.

3.5.4 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(see appendix), 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, it would be more efficient and convenient to use Django's Object Relation Mapping capabilities.



UserManager
PermissionMixin
MyUserManager
attributes
#email_prompt: BooleanField
operations
#create_user(email: String, password: String = None, **extra_fields) : None
#create_superuser(email: String, password: String = None, **extra_fields) : None
AbstractBaseUser
MyUser
attributes
+user_id: AutoField
+email: EmailField
+is_staff: BooleanField
+creation_date: DateField
+first_name: CharField
+last_name: CharField
+USERNAME_FIELD: String
+REQUIRED_FIELDS: List[String]
operations
+get_full_name() : String
#set_password(raw_password: String) : None
#check_password(raw_password: String) : Bool
One to one
Association
1
has
1
Entry
attributes
+entry_id: AutoField
+journal_id: ForeignKey[Journal]
+creation_date: DateField
+title: CharField
+content: TextField
+sentiment: FloatField
operations
+__str__() : String
- analyse_sentiment() : float
Journal
attributes
+journal_id: AutoField
+user_id: OneToOneField[MyUser]
+creation_date: DateField
operations
+get_all_entries() : QuerySet
+__str__() : String
models.Model

Figure 3.5: UML Class Diagram

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

- MyUser: A class which 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 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 capabilities 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.

3.6 File / Table Structures

Explain the file or table structures used.

3.7 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 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.

Sorting algorithms like bubble sort and insertion sort are bad for large datasets since they have a time complexity of O(n^2) on average. Hence, I will be choosing an efficient sorting algorithm such as Quicksort or Merge Sort, two divide and conquer algorithms, which have a time complexity of $O(n \log n)$ on average. Between the two, both offer good performance but they have different implementations and trade offs. I have chosen Quicksort over merge due to the fact that it is In-Place. This means that copies of the list are not created to sort it, making it more efficient in terms of memory.

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 of the creation date of the entry.

3.8 User Interface

3.8.1 Wireframe Design

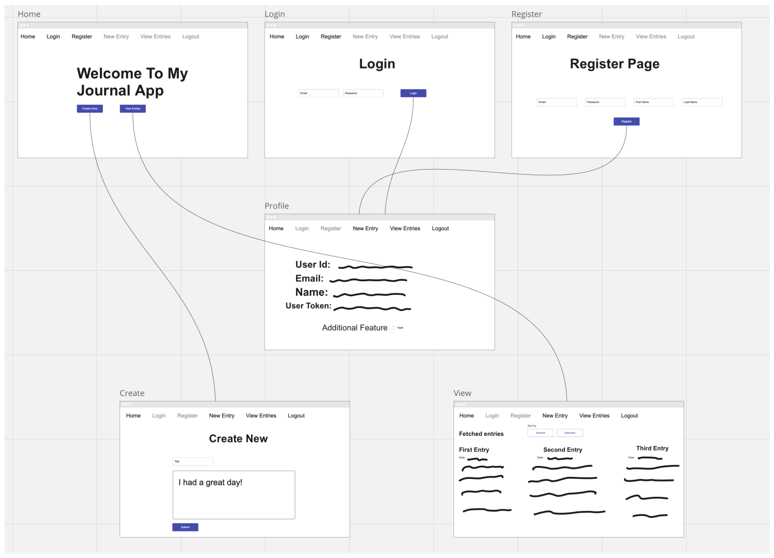


Figure 3.6: This is an overview of all the screens inside my application, it includes the home page, login page, register page, profile page, create page and view page. There are lines connecting the pages to show the flow of the application. Using the navigation bar at the top, the user can navigate to any page from any page.

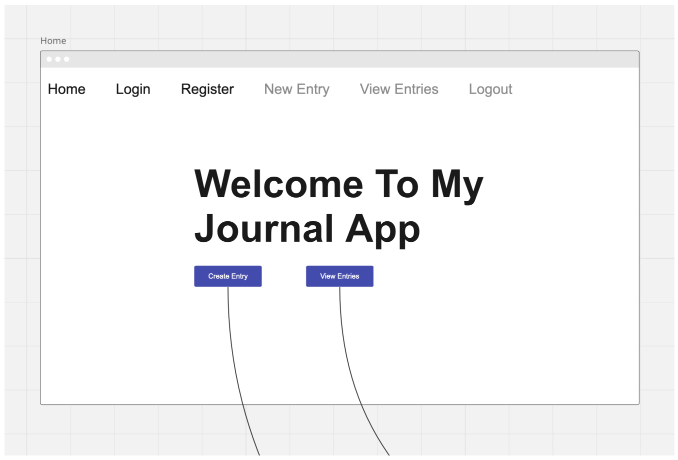


Figure 3.7: The homepage of my application, it includes a navigation bar at the top, a welcome message and buttons for creating and viewing the user's journal entries.

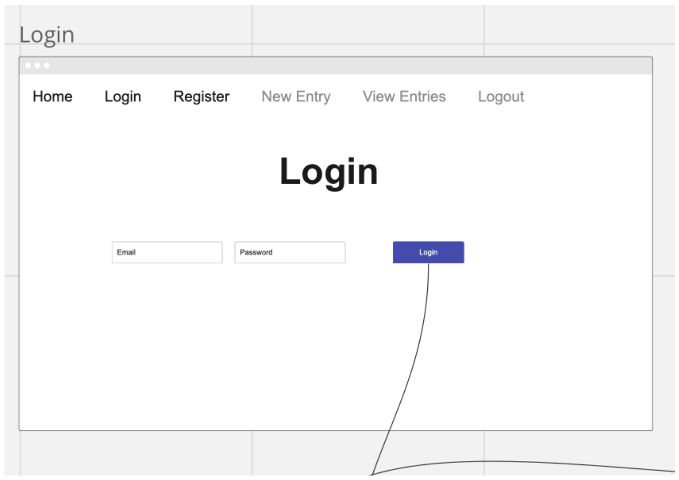


Figure 3.8: The login page of my application, it includes a navigation bar at the top, a form for the user to input their email and password and a button to submit the form. As you can see in the navigation bar, New Entry, View Entries and Logout are all greyed out, this is assuming that the user is not logged in.

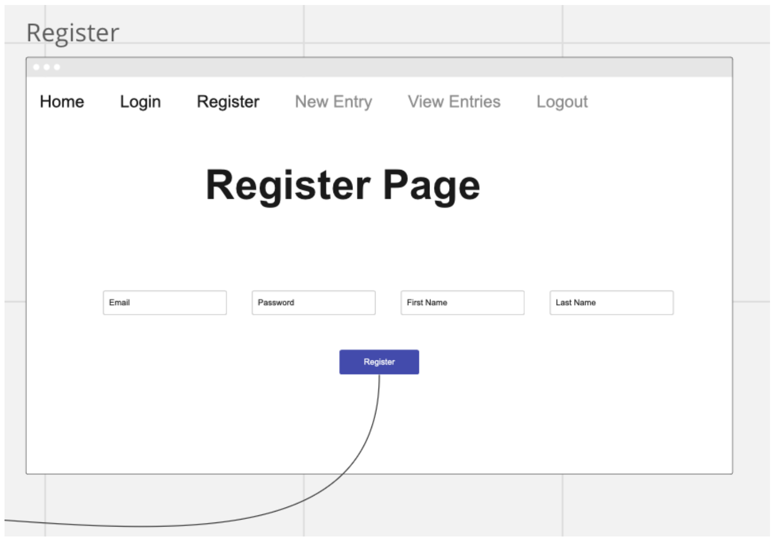


Figure 3.9: Register page is very similar to the login page, it includes a navigation bar at the top, a form for the user inputs as well as the submission button. Registration field requires a couple more fields than the login page.

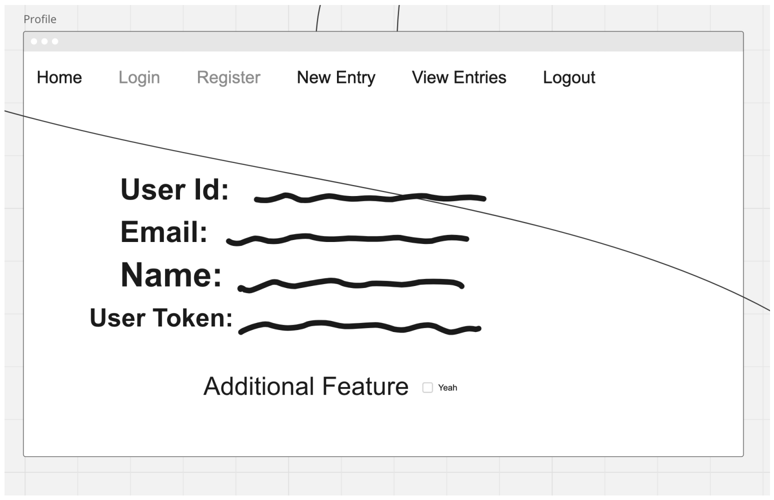


Figure 3.10: Profile page is where the user can view their profile information, as well as place where they can opt in for some additional features.

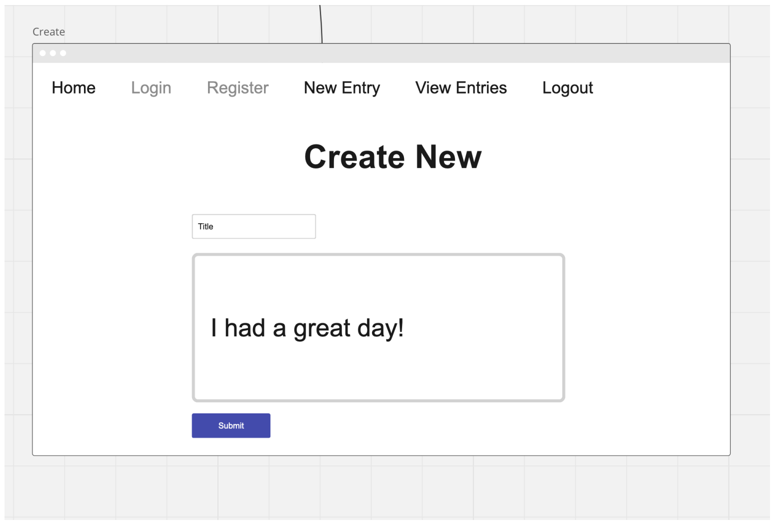


Figure 3.11: Create page is where the user can create a new journal entry. It includes a form for the user to input the title and content of the entry.

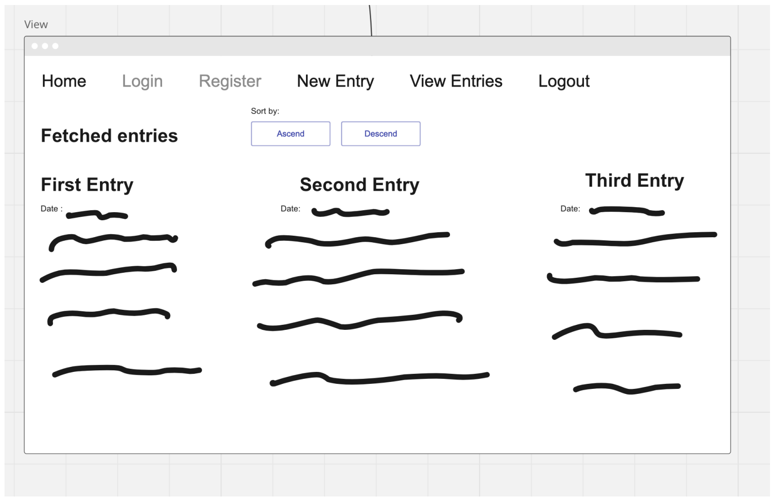


Figure 3.12: View page is where the user can view all of their journal entries. It includes buttons to enable users to sort the entries.

3.9 RESTful API Endpoints

3.10 Hardware & Software Requirements

Specify the hardware and software requirements for the project.

3.11 User Authentication

3.11.1 Session Based Authentication

Session based authentication is where a session is generated and an ID is stored in the Cookie

Chapter 4

Technical Implementation

4.1 Key Code Segments

Highlight essential code segments.

4.2 Data Structures

Explain the implementation of data structures.

4.3 Algorithms

Detail the implementation of algorithms.

4.4 Modularity

4.4.1 Virtual Environment

Discuss the modularity of the project.

4.5 Defensive Programming / Robustness

Explain the measures taken for defensive programming and robustness.

Cross Site Request Forgery (CSRF)

4.5.1 Server Based Authentication

Cross-Site Scripting (XSS)

4.6 Patterns

4.6.1 Single Responsibility Principal

4.7 Credential Management

Chapter 5

Testing

5.1 Test Strategy

Describe the overall testing strategy.

5.1.1 Rest Client

5.1.2 Django Test Models

5.2 Testing Video

Include a link or description of a testing video.

5.3 System Tests

Conduct tests against the original requirements specification.

Chapter 6

Evaluation

6.1 Requirements Specification Evaluation

Evaluate the project against the initial requirements.

6.2 Independent End-User Feedback

Include feedback from end-users.

6.3 Improvements

Discuss potential improvements for the project.

Chapter 7

Appendix

7.1 Code Listing

Include relevant code listings.

7.1.1 Appendix A – Code Listing

Include code listings as an appendix.