SOFTWARE ENGINEERING PROJECT DOCUMENT

B.TECH. (CSE) V SEMESTER
UE20CS303 –SOFTWARE ENGINEERING
PROJECT REPORT ON

GARBLE: An Encrypted Journal

Team Details:

1. Keerthi R - PES1UG20CS205
2. Maani Jacob Manayath - PES1UG20CS241
3. Mahesh KM - PES1UG20CS242
4. Mahika Gupta - PES1UG20CS243

August – Nov 2022
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SOFTWARE ENGINEERING PROJECT DOCUMENT

BENGALURU – 560100, KARNATAKA, INDIA

TABLE OF CONTENTS

Proposal of the Project

Proposed Project Description:

A journal in which entries made can be encrypted to maintain confidentiality.

Our target audience consists of journalists, writers, regular travelers, youngsters who want to preserve their privacy, professionals who want to protect their intellectual property, and entrepreneurs who seek to store and keep track of their ideas.

Features include:

- The journal entries are encrypted.
- Password protection.
- The program also analyzes the tags of the entries, based on which the user's moods on any particular day are tracked.
- Local database to back up the data for ease of access in the user’s devices.

Plan of Work and Product Ownership:

In the next few weeks, we plan to:

- Create a structured prototype of our web application. This will be done by Keerthi.
- Basic functionality to enter the journal entries. This will be done by Maani and Mahesh.
- Login using a password, for the user to access his entries, done by Mahika.
- Look into the encryption methods to be used to encrypt the journal entries, done by Mahika

Table of Contents:

Revision History

Name	Date	Reason For Changes	Version
Garble: An encrypted journal	**07/09/20 22**	**-**	**1.0**

SOFTWARE REQUIREMENTS SPECIFICATION

Introduction

1.1 Purpose:

Garble is an application in which journal entries made can be encrypted to maintain confidentiality. The program will also track the user's mood based on the entries made. This document covers the requirements of the first completed model of the application.

1.2 Intended Audience:

The target audience consists of journalists, writers, regular travelers, youngsters who want to preserve their privacy, professionals who want to protect their intellectual property, and entrepreneurs who seek to store and keep track of their ideas. This document includes the product scope, description, interface requirements, analysis models, system requirements, and functional and non-functional requirements.

1.3 Product Scope:

This web application will let the user make journal entries. The user can log in using a login id and password. This will ensure that only authorized users can access the journal. The entries made will be encrypted using an encryption algorithm(AES). This will keep it confidential. For every entry made, a trained machine-learning model will predict the mood of the user. Using this feature the user can keep track of his/her mood. A local database will be maintained to keep the journal entries of the user backed up, for ease of access on their devices. This application can be used by anyone using a browser application, who wants to keep their information confidential.

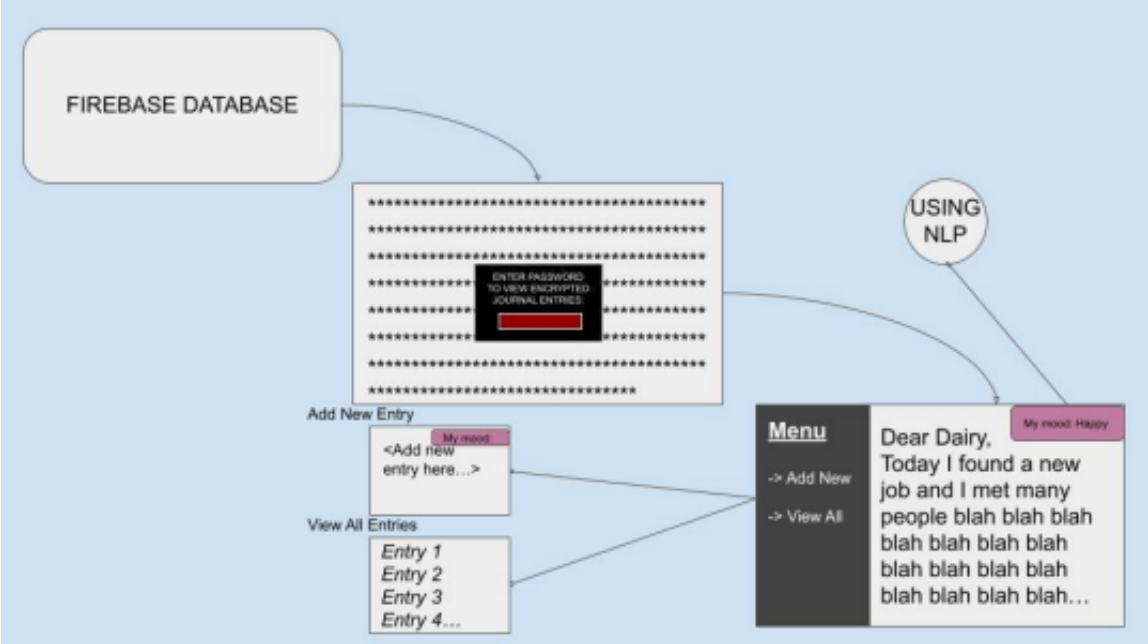
1.4 References :

Project Synopsis : ProjectSynopsis_team5

Overall Description

2.1 Product Perspective:

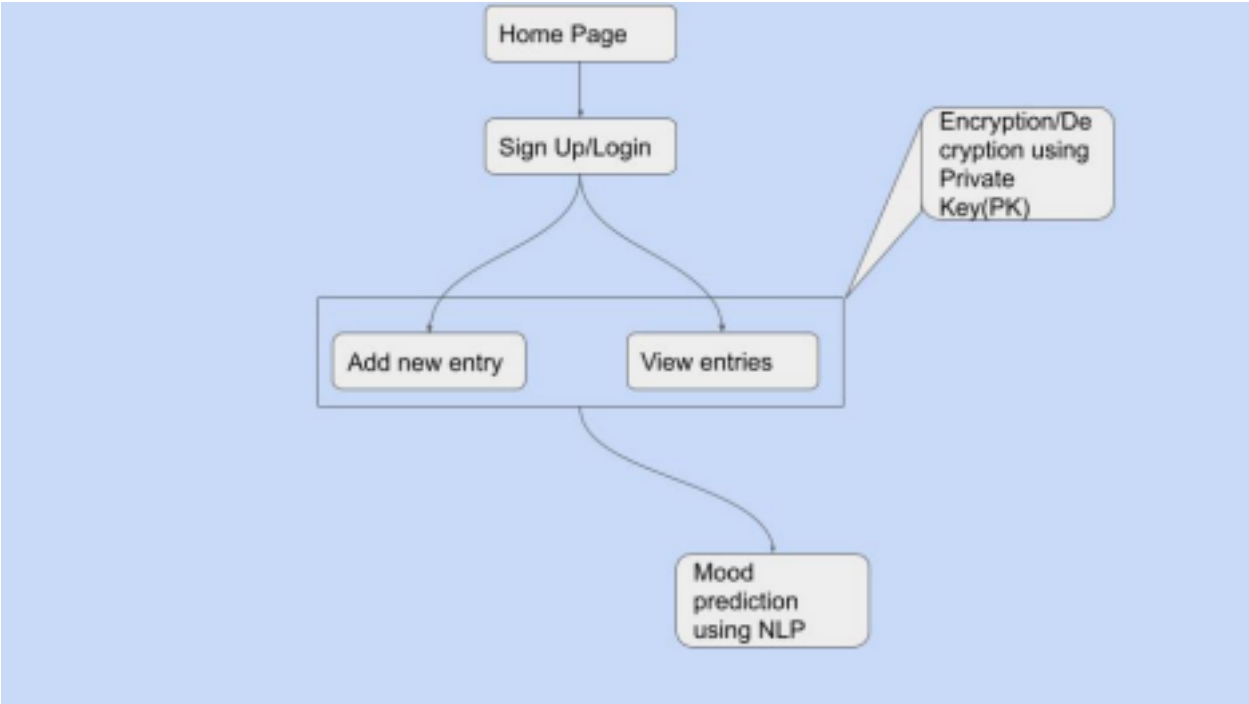
This application is a new, self-contained product. It is open-source. A local database will be maintained on TinyDB, and the data in the database can be accessed at any time quickly as it is stored on the local host machine.



2.2 Product Functions :

Functions:

- Logging in using email id and password
- Making a new journal entry
- sorting journal entries in chronological order (based on the date of entry)
- encrypting and decrypting journal entries using a private key
- mood predicting to track mood every day based on journal entry tag



2.3 User Classes and Characteristics

1. Technical Professionals- intellectuals who want to protect
   their information: - researchers
   - professors with theses
   - scientists
   - for protection of intellectual property
2. Journalists- can make entries anytime, anywhere:
   - Travelers
   - bloggers
   - writers
3. Students/youth - can keep their entries private and track their daily mood:
   (most important class to satisfy)
   - Teenagers
   - diarists
   - individuals subjected to mental illnesses

2.4 Operating Environment

1. System requirements:
    a. minimum 2GB memory
    b. processor - 2 x 1,6 GHz CPU (recommended)
    c. Windows 10 and above/MacOs/Linux 64-bit
    d. Storage minimum 2GB
2. Software requirements:
    a. VS code editor
    b. TinyDB database
    c. web browser - Google Chrome, Firefox, Safari, Brave, etc
    d. Programming languages:
        i. Python
        ii. Flask
        iv. HTML
        v. CSS

2.5 Design and Implementation Constraints:

The goal of the application is to be a native app on the client side. Therefore the code will have to be implemented from the client side. Some other limitations are:

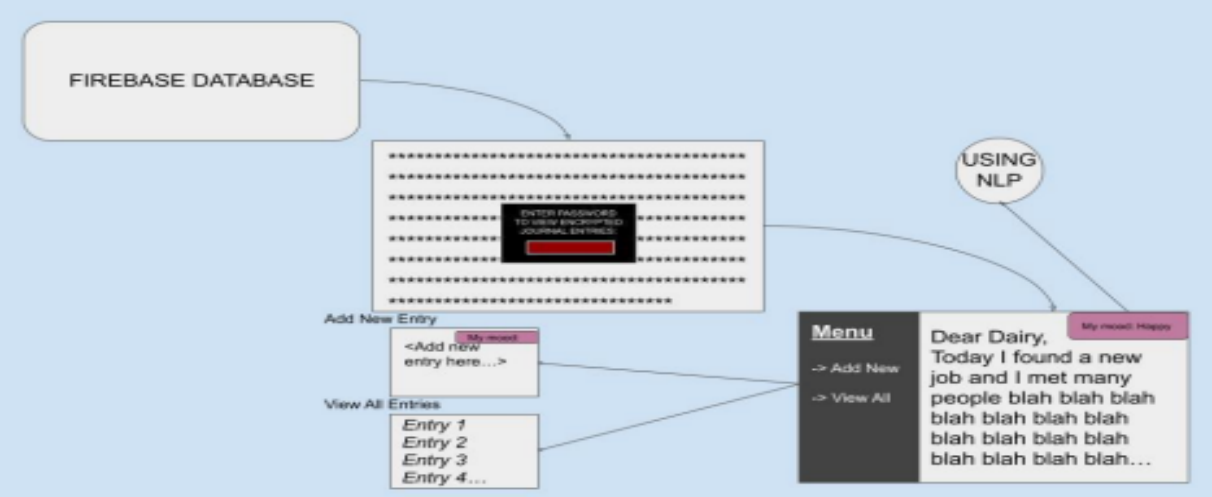
- Separate code base must be created and maintained for each individual platform.
- Customers cannot upload images, videos, or any other media to the journal entries as input.
- Security - user will have to keep his/her login password and key safe.
- Application limited to the host's machine.
- The data stored in the database should be accessible by the application.

2.6 Assumptions and Dependencies

- It is assumed that the TinyDB software will work correctly with the application program and that there is sufficient storage.
- If TinyDB database fails to handle our traffic well, we will have to explore other database management systems.
- Application requires Python, Flask, and TinyDB to be developed.

External Interface Requirements

3.1 User Interfaces :



The font used in our app would be “Open Sans”. The color palate that is used for the user interface will be of minimalist theme with mild colors like grey and white. However, error messages and dialogue boxes will be in darker shades. We will be using CSS stylesheets for the UI and similar style guides will be followed. As you can see in the above image, we will have a bar with “Menu” which will redirect us to either a “Add new entry” or “View all entry”. The “Mood” of the writer which will use NLP for the prediction will be displayed on the top right corner of every entry.

3.2 Software Interfaces

This software will be based on the TinyDB database(v4.7.0), Python (v3.9.8), and Flask (v2.2.2) for the entire development of the software. The entire software will be highly portable and will be run on all operating systems (Windows/MacOS/Linux). The data that is stored in the software would be local to the user. Login and password protected by encryption and decryption of data using a private key and AES encryption scheme implemented in python.

3.3 Communications Interfaces

The software does not support any sort of communication within the application.

Analysis Models

Database - Schema Diagram

User

U_I	Name	Email	Ph. No.

Mood(Weak Entity)

U_I	E_I

EntriesD

E_ID	Date	Name	Description

System Features:

Functional requirements:

- Login authorization is done using TinyDB.
- Encryption of the text using the AES algorithm.
- Database local to back up the data entered in the Journal.

Major services :

- Maintains privacy and confidentiality.
- Enables the users for effortless storage of ideas.
- Secure storage of data.
- Information gets encrypted before being saved.

1. Data encryption:

5.1.1 Description and Priority

.**Data encryption:** All the data entered in the journal shall be encrypted before being saved.

Priority: High
Benefit: 8
Penalty: 4
Cost: 6
Risk: 2

5.1.2 Stimulus/Response Sequences

- User enters Data / Information into a journal.
- When the user wants to save the data entered, we encrypt the data

and store it in a locally hosted database.
- Data is encrypted using the AES algorithm.

5.1.3 Functional Requirements

REQ-1: Up-to-date version of Python
REQ-2: Any common browser software to host app natively
REQ-3: AES algorithm python libraries: An algorithm which encrypts data

2. Password Protection:

5.1.1 Description and Priority

Password protection: All the data entered in the journal shall be encrypted before saving it. Passwords stored in the database are hashed for privacy.

Priority: High
Benefit: 9
Penalty: 3
Cost: 7
Risk: 2

5.1.2 Stimulus/Response Sequences

- Separate account is created for each user which is password protected - This enables the user to keep his data secure.

5.1.3 Functional Requirements

REQ-1: Up-to-date version of python
REQ-2: Python cryptography libraries
REQ-3: SHA-256 algorithm

3. Database:

5.1.1 Description and Priority

Database: This feature makes a backup store of all the user's data and stores it in a locally hosted database.

Priority: High
Benefit: 8
Penalty: 4

Cost: 9
Risk: 4

5.1.2 Stimulus/Response Sequences

- After the data is entered in the journal, the user will save the data
- We save the data in a locally-hosted database.
- It will be easy to modify the data whenever the user wants to.
- Data can be retrieved quickly and with ease.

5.1.3 Functional Requirements

REQ-1: TinyDB database
REQ-2: Python (v3.9.8)

Other Nonfunctional Requirements

6.1 Performance Requirements

- CLOCK SPEED: 2.8 GHz
- MAIN MEMORY : 1GB
- SECONDARY MEMORY : 40GB
- CACHE MEMORY : 1MB
- ERROR RATE: The average number of problem requests compared to total requests is your error rate.

6.2 Safety Requirements

- Safeguarding of key: Information and diary entries may be leaked to an unauthorized individual/organization if the key is not safely guarded.
- Input validation: lack of which may trigger malfunction.
- Secure data storage.
- Loss of data due to failure in technology possible.

6.3 Security Requirements

- Authentication: verify client's identity before giving access.
- Personal information such as email id and phone number may be leaked to unauthorized individuals/organizations if https or secure web access is not used.
- Use encryption

6.4 Software Quality Attributes

- This journal can be used by everyone for many purposes
- All the features provided are secure and easily adaptable.

- The Journal is flexible to be used by people from different professions.
- It's a native web application, which makes it much simpler for the user to edit and access it whenever he wants, quickly.
- This is a reusable and user-friendly native web application.

6.5 Business Rules

A person who likes to digitally secure their journal entries and at the same time monitor their mood over a period of time and gain a better understanding of themselves would use this software. There are no other rules that are supported in this software.

Other Requirements

None that have not already been mentioned.

Appendix A: Glossary

- AES - Advanced Encryption Standard
- HTML - Hypertext Markup Language
- HTTPS - HyperText Transfer Protocol Secure
- SDK - Software Development Kit
- SSH - Secure Shell

Appendix B: Field Layouts

An Excel sheet containing field layouts and properties/attributes and report

requirements.

Appendix C: Requirement Traceability Matrix

Sl No	Requirement ID	Brief Description of Requirement	Architecture Reference	Design Reference	Code File Reference	Test Case ID	System Test Case ID
1	01	Login page - to give access to		https://assets.justin		11	21

		user to his/her journal		mind.com /wp-conte nt/upload s/2018/10 /diprella_l ogin.gif			
2	02	Password protection - to authorize user		https://ass ets.justin mind.com /wp-conte nt/upload s/2018/10 /diprella_l ogin.gif		12	22
3	03	Encryption algorithm - AES	https://d3i71x aburhd42.clou dfront.net/52f a071b34d9e3f 2573e59e1d6 71cad6708a36 1f/2-Figure2-1 .png		https://www .delftstack.c om/howto/p ython/pytho n-aes-encryp tion/#:~:text =decrypt%20 the%20mess age.-,the%20 AES%20256 %20Using%2 0PyCrypto% 20in%20Pyth on,in%20fun ctionalities% 20related%2 0to%20crypt ography%20.&text= Block %20size%20i s%20set%20 to,by%20app ending%20s ome%20addi tional%20byt es.	13	23
4	04	Database	https://www.re			14	24

		handling - using TinyDB	searchgate.net/profile/Farrukh_Khan5/publication/224969017/figure/fig1/AS:302667139174400@1449172877064/Basic-Architecture-of-TinyDB.png				

PROJECT PLAN

Project Plan Document

Lifecycle to be followed for the execution of the project:

Incremental cycle.

We want to make each deliverable one at a time, and increment the project with new functionalities.

For each increment, we will be using the 'Waterfall model'.

Incremental models are easier to test and debug than smaller modules.

The incremental module reduces functionality and gives priority to essential features over additional ones.

We want to release a working version of the software with every module.

Tools used throughout the lifecycle:

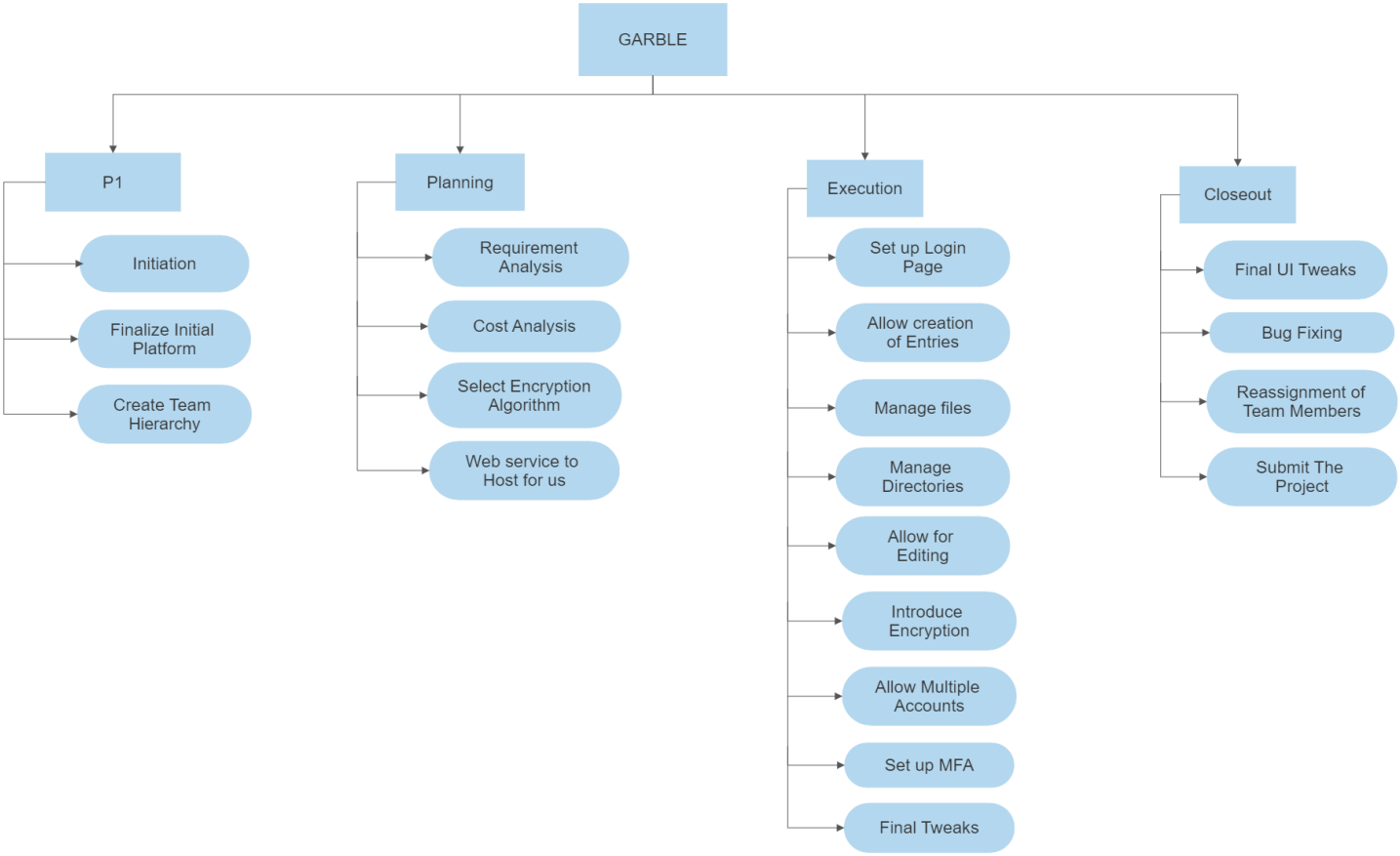
1. Planning Tool: Atlassian Jira
2. Design Tool: Canva
3. Version control tool: GitHub
4. Development tool: Visual Studio Code, Jupyter Notebook
5. Bug tracking: Sentry
6. Testing Tool: Selenium

Deliverables:

1. Login page: username and password: Build Component
2. Creating new entry: Build Component
3. Storing: Build Component
4. Viewing/opening: Build Component
5. Editing: Reuse Component: Can reuse the functionality of the "Create New Entry" feature
6. Encrypting: Build Component
7. Mood prediction: Build Component
8. Multiple account login: Reuse Component - Can reuse the functionality of the "Login" feature

Work Breakdown Structure:

P1							0%
Initiation		100%	Me...			No	0%
Finalize initial platform		100%	Me...			No	0%
Create team hierarchy		100%	Me...			No	0%
Planning							0%
Requirement Analysis		100%	Me...			No	0%
Cost Analysis		100%	Me...			No	0%
Select Encryption Algorithm		100%	Me...			No	0%
Web Service to host for us		100%	Me...			No	0%
execution							0%
Set up Login page : userna...		100%	Me...			No	0%
Allow creation of entries		100%	Me...			No	0%
Manage storing the files, fi...		100%	Me...			No	0%
Manage opening files dire...		100%	Me...			No	0%
Allow for editing of entries		100%	Me...			No	0%
Introduce encryption of e...		100%	Me...			No	0%
allow multiple accounts to...		100%	Me...			No	0%
Set up MFA		100%	Me...			No	0%
Final tweaks		100%	Me...			No	0%
Closeout							0%
Final UI tweaks		100%	Me...			No	0%
Bug fixing		100%	Me...			No	0%
Reassignment of team me...		100%	Me...			No	0%
Submit the project.		100%	Me...			No	0%



Estimate of effort required:

- 1 week: Planning - 4 people
- 1 week : Create a design of our web application - 2 teammates.
- 1 weeks : Login using a password, for the user to access his account - 2 teammates.
- 1 week: Basic functionality to enter the journal entries . - 2 teammates
- 1 week: Editing feature - 2 teammates
- 1 week : Look into the encryption methods to be used to encrypt the journal entries - 2 teammates.
- 2 weeks: Create the encryption algorithm program to encrypt entries. - 3 teammates
- 1 week: Configuring the storage tool with the program application. - 2 teammates
- 2 weeks: Creating, training and testing the mood prediction machine learning model. - 4
- 2 weeks: Additional features (multiple login, multi factor authentication) - 4 teammates.

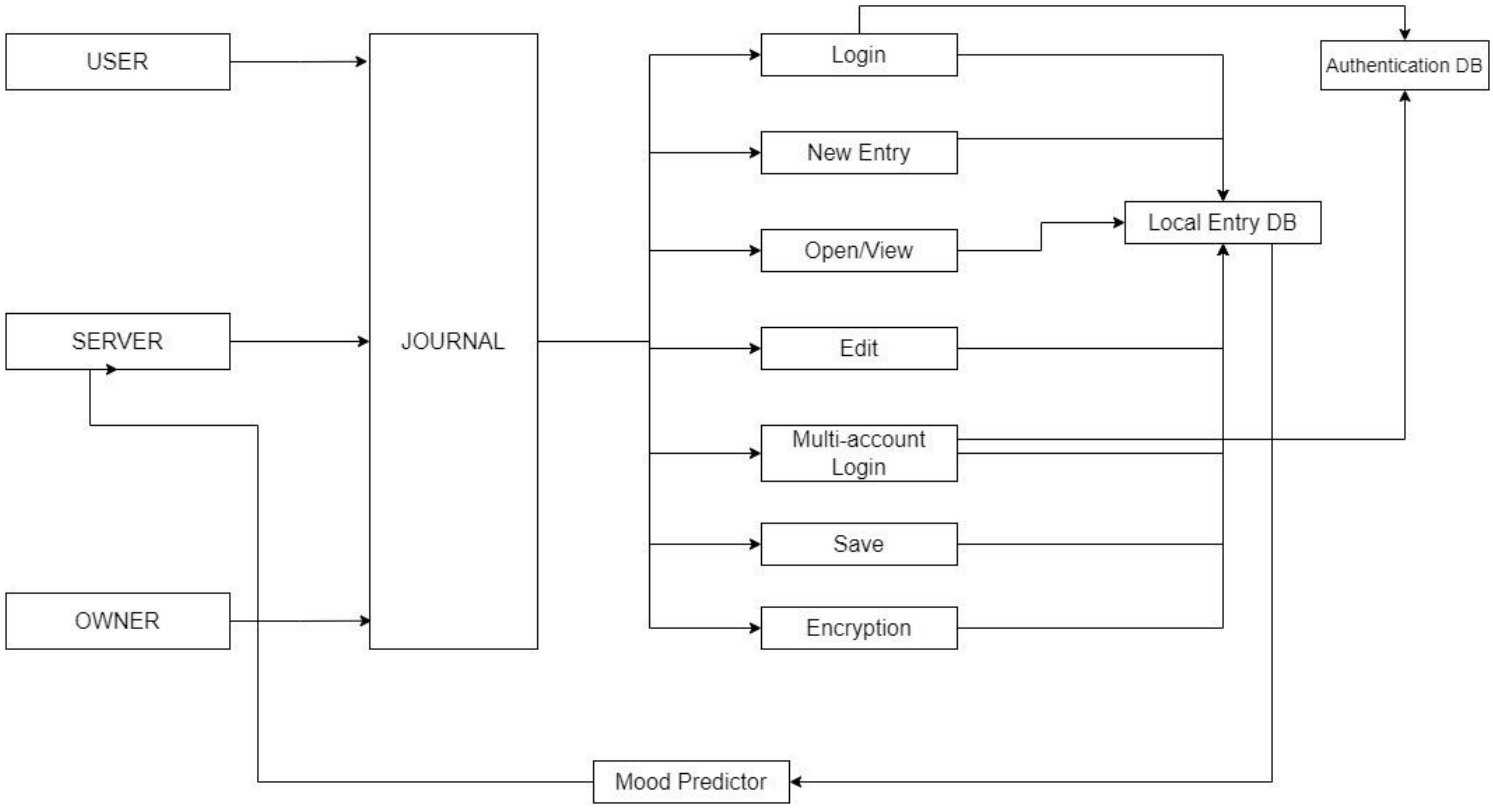
Create the Gantt Chart for scheduling using any tool.

GARBLE - GANTT CHART

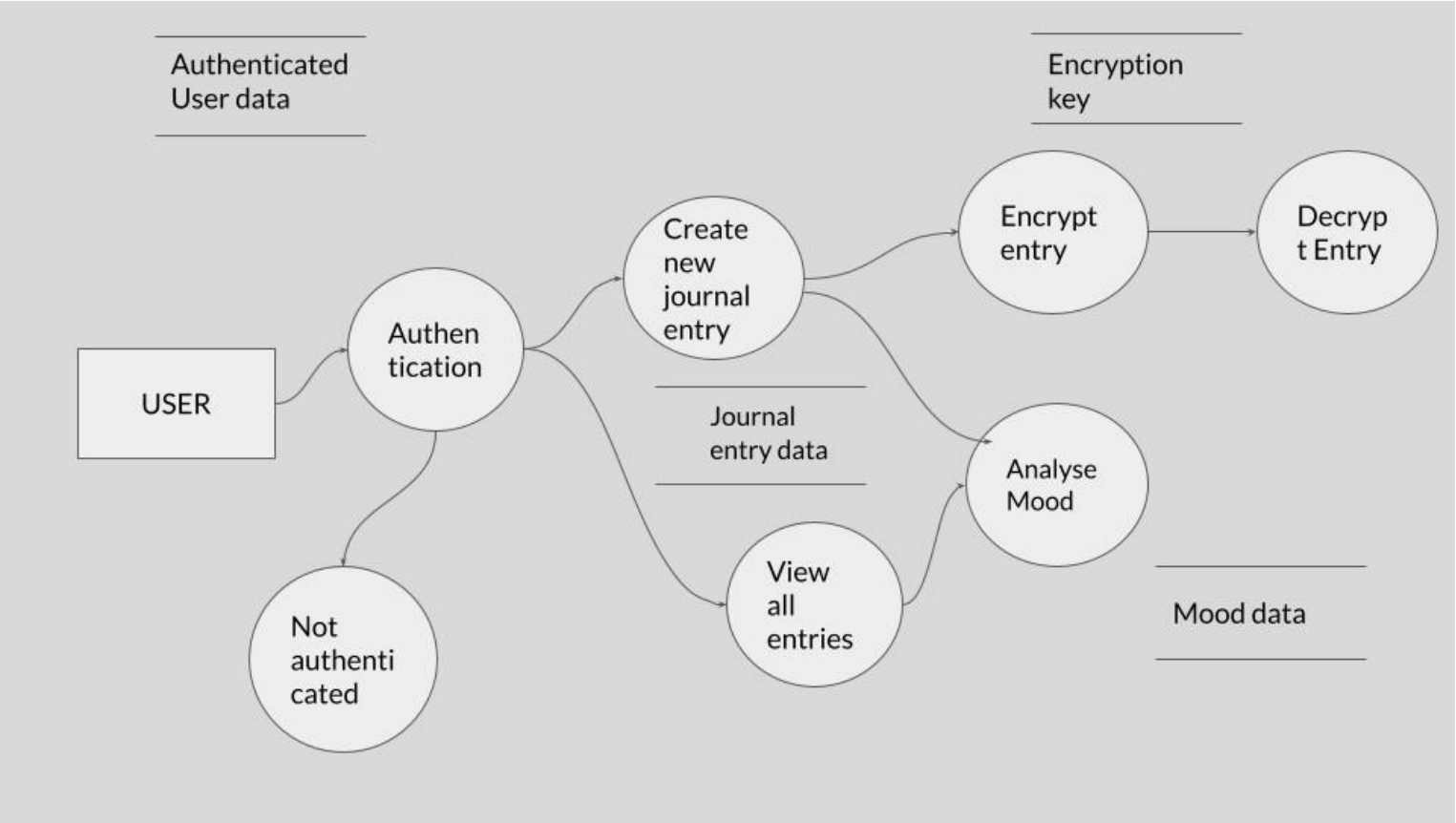
DESIGN DIAGRAMS

Propose Architectural style /design for the project selected. Design pattern for their problem statement(Architectural) identifying Quality Attributes

Architectural design:



Data Flow Diagram:



CODE SEGMENT SCREENSHOTS

1. Python-Flask module for frontend

run.py

```python
from app import app


# Run app
if __name__ == '__main__':
    app.run(debug=app.config['DEBUG'])
```

forms.py M

app > admin > forms.py > ...

```python
from flask_wtf import Form
from wtforms import StringField, SubmitField
from wtforms.validators import Required


class RenameChronofileForm(Form):
    new_name = StringField('Enter new name for journal:', \
                           validators=[Required()])
    submit = SubmitField('Rename journal')


class RenameAuthorForm(Form):
    new_name = StringField('Enter new author name:', \
                           validators=[Required()])
    submit = SubmitField('Rename author')
```

pagination.py

app > pagination.py > get_entries_for_page

```python
from flask import redirect, url_for, session, abort
import ujson
from app.db import *


def update_pagination():
    ''' Creates a table of entries organized by page
    for use when browsing entries. '''
    results = search_records('entries', \
        Query().creator_id == session.get('user_id'))
    all_entries = results[::-1]
    limit = len(all_entries)
    total = 0 # counter for total number of entries processed
    count = 0 # counter for number of entries-loop at 10
    page = 1 # will be key for dictionary of pages
    p_entries = list()
    get_table('pagination').purge() # clear the old pagination table
    for entry in all_entries:
        total += 1
        count += 1
        p_entries.append(entry)
        if count == 10 or total == limit:
            insert_record('pagination', {'page': page, 'entries': p_entries})
            del p_entries[:] # start the list fresh
            page += 1
            count = 0
    return True


# Get entries for the given page
def get_entries_for_page(page):
    results = get_record('pagination', Query().page == page)
    if not results:
        return abort(404)
    else:
        return results['entries']

def check_next_page(page):
    next_page = page + 1
    if not get_record('pagination', Query().page == next_page):
        return None
    else:
        return next_page
```

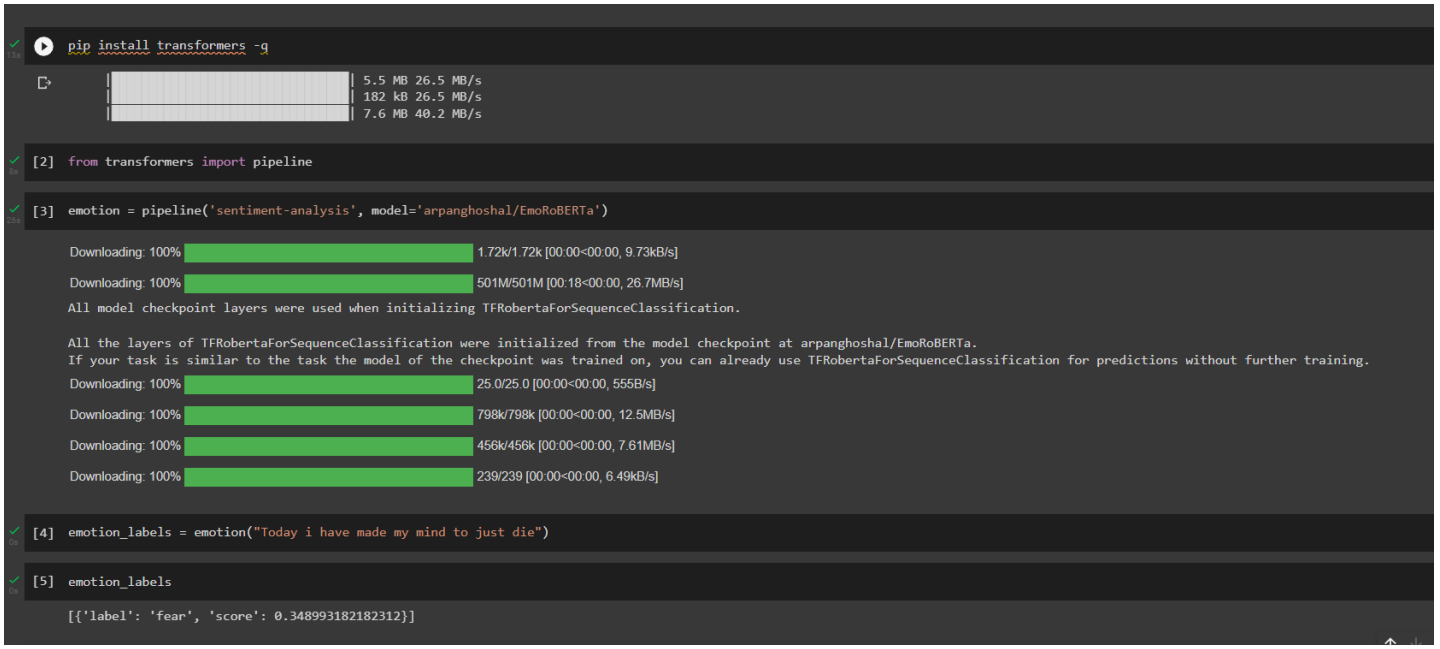
2. AES journal encrypting module:

aes.py

AES-Enc > aes.py > data

```python
from Crypto.Cipher import AES

def acceptKey():
    key_string = input("Enter key of 16 characters:")
    if len(key_string)!=16:
        raise Exception("Key length not 16")
    key = bytes(key_string, 'utf-8')
    return key

def encryptText(key,data):
    cipher = AES.new(key,AES.MODE_EAX)
    nonce = cipher.nonce
    ciphertext = cipher.encrypt(data)
    return ciphertext,nonce

def decrypt(key,ciphertext,nonce):
    cipher = AES.new(key,AES.MODE_EAX,nonce=nonce)
    plaintext = cipher.decrypt(ciphertext).decode()
    return plaintext


data = input("Enter diary entry:").encode()
key = acceptKey()
ct,nonce = encryptText(key,data)
pt = decrypt(key,ct,nonce)
print("Encrypted text:", ct)
print("Decrypted text:", pt)
```

3. NLTK module for sentiment analysis



4. Database handling module:

db.py

app > db.py > search_records

```python
from tinydb import TinyDB, Query
import ujson


def get_db():
    db = TinyDB('db.json')
    return db


def get_table(table_name):
    table = get_db().table(table_name)
    return table


def get_record(table_name, query):
    result = get_table(table_name).get(query)
    return result


def search_records(table_name, query):
    results = get_table(table_name).search(query)
    return results


def insert_record(table_name, record):
    # Inserting a record returns the element id of the new record.
    element_id = get_table(table_name).insert(record)
    return element_id


def update_record(table_name, field, query):
    get_table(table_name).update(field, query)
    return True


def get_element_id(table_name, query):
    element = get_record(table_name, query)
    element_id = element.eid
    return element_id
```

User Story

Our target user is a creative person or professional who wants to maintain a certai level of personal security when it comes to their written work. That person does not wish to use a heavy application for this purpose and would instead require a lightweight solution.

Garble and its encrypted journal tracking system makes it easy for any creative person to easily save and order their written work.

Let us take the example of Kingshuk, an aspiring writer who writes down his ideas and plans for new books on his laptop. His current job requires him to share his laptop with multiple other colleagues and thus he finds it difficult to maintain his privacy and the security of his intellectual property.

With Garble, he can create encrypted, well catalogued journal entries that nobody apart from him can access. He is able to keep track of what ideas he had on which day and even tag his work so that he can sort it by genre.

His coworkers do not have the key to the encryption and thus his privacy is guaranteed. The lightweight design and the pleasing user interface further exemplifies the benefits of the product.

This product can also be used by employees to encrypt sensitive information and by innovators to secure new ideas.

TESTING

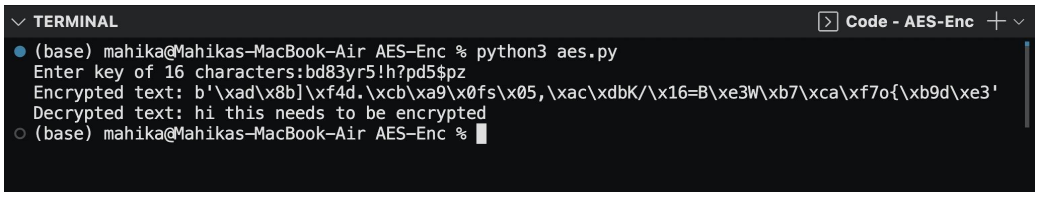
Unit Testing

A unit is the smallest block of code that functions individually. The first level of testing is Unit testing and this problem statement is geared towards the same.

- Discuss with your teammates and demarcate units in your code base
  Note: discuss why the code snippet you have chosen can be classified as a unit
  - The following part of the code base is chosen as a unit for unit testing, as it is a functionality which is independent of other units of the code, and can be tested easily. The following

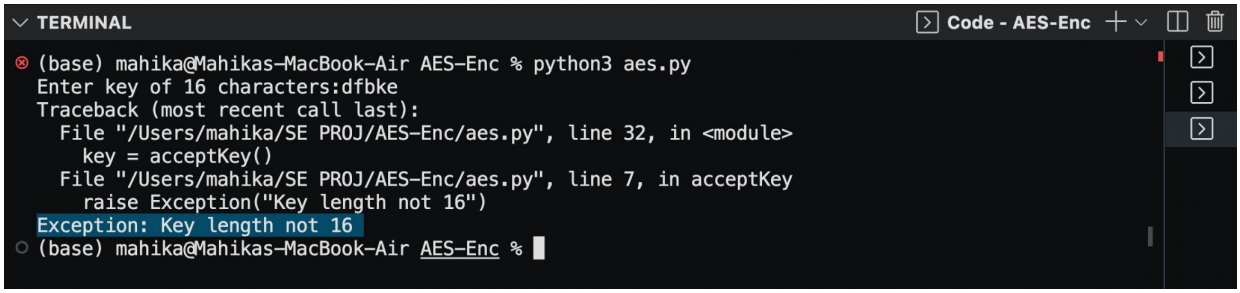
```python
from Crypto.Cipher import AES


def acceptKey():
    key_string = input("Enter key of 16 characters:")
    if len(key_string)!=16:
        raise Exception("Key length not 16")
    key = bytes(key_string, 'utf-8')
    #print(key)
    return key

#mahika
# key= b'C&F)H@McQfTjWnZr'
def encryptText(key):
    #key = acceptKey()
    cipher = AES.new(key,AES.MODE_EAX)
    nonce = cipher.nonce
    data = "hi this needs to be encrypted".encode()
    ciphertext = cipher.encrypt(data)
    #print("Ciphertext:",ciphertext)
    return ciphertext,nonce

def decrypt(key,ciphertext,nonce):
    # key = acceptKey()
    # ciphertext,nonce = encryptText()
    cipher = AES.new(key,AES.MODE_EAX,nonce=nonce)
    plaintext = cipher.decrypt(ciphertext).decode()
    #print("Plaintext:",plaintext)
    return plaintext


key = acceptKey()
ct,nonce = encryptText(key)
pt = decrypt(key,ct,nonce)
print("Encrypted text:", ct)
print("Decrypted text:", pt)
```

- Develop test cases for both valid and invalid data
  - Test case 1: for valid data:



    Here since the constraint on the key input is to enter a key with 16 characters, the test case passes as a 16 byte key is entered. Test case is passed.

  - Test case 2: for invalid data



  - Here we see that the condition for the key size being 16 characters is not satisfied. The value entered is invalid and an exception is raised. Test case is passed.
- Ideate how you could further modularize larger blocks of code into compact units with your teammates
- This unit can be further isolated for rigorous testing by dividing it into 2 units of one function each:
  - One unit for the function encryptText(key):
  - The other unit for the function decrypt(key,ciphertext,nonce)

Dynamic Testing

Dynamic testing involves the execution of your code to analyze errors found during execution. Some common techniques are Boundary Value Analysis and Mutation Testing.

Boundary Value Analysis

When it comes to finding errors in your code base, they are often found at locations where a condition is being tested. Due to this, developers often use Boundary Value tests to reduce defect density.

- How would you define a boundary test?

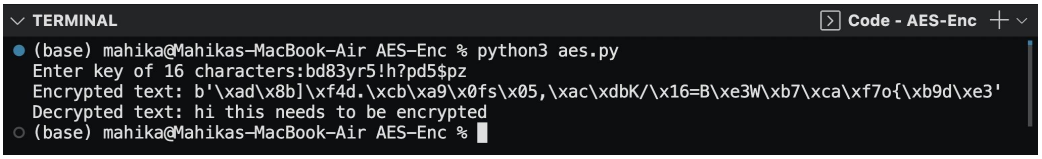
Note: Simple relational conditions are a basic example

Boundary testing is a black-box testing technique used to identify the errors at the boundary or the extreme ends of the model.

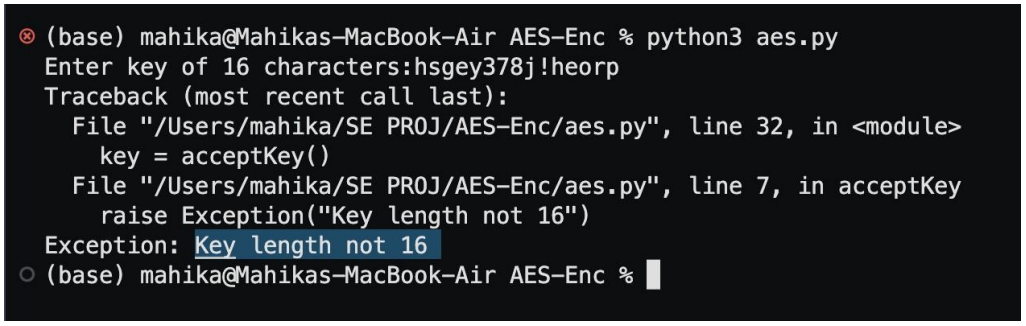
- Build your boundary test cases and execute them

In the unit code that we are testing, the input must have exactly 16 characters. Hence, in this case the boundary value considered will be 16.

- Test 1: entering boundary value as input: On passing an input of 16 characters:



- Test 2: entering value less than boundary value: On passing a value with 15 characters:



- Test 3: entering value greater than boundary value: On passing a value of 17 characters:

```
(base) mahika@Mahikas-MacBook-Air AES-Enc % python3 aes.py
Enter key of 16 characters:hu8uie0?j2#ywh0ks
Traceback (most recent call last):
  File "/Users/mahika/SE PROJ/AES-Enc/aes.py", line 32, in <module>
    key = acceptKey()
  File "/Users/mahika/SE PROJ/AES-Enc/aes.py", line 7, in acceptKey
    raise Exception("Key length not 16")
Exception: Key length not 16
```

Mutation Testing

- Using your isolated units from the first problem statement, ideate with your teammates on how to mutate the code

The code can be mutated such that key of length greater than 15 is accepted (still throws an error due to python packages)

Original code:

```
def acceptKey():
    key_string = input("Enter key of 16 characters:")
    if len(key_string)!=16:
        raise Exception("Key length not 16")
    key = bytes(key_string, 'utf-8')
    #print(key)
    return key
```

Mutated code:

```
def acceptKey():
    key_string = input("Enter key of 16 characters:")
    if len(key_string)>15:
        raise Exception("Key length not 16")
    key = bytes(key_string, 'utf-8')
    #print(key)
    return key
```

Output with error:

```
(base) mahika@Mahikas-MacBook-Air AES-Enc % python3 caseStudy.py
Enter key of 16 characters:yueripshdjfkith76
Traceback (most recent call last):
  File "/Users/mahika/SE PROJ/AES-Enc/caseStudy.py", line 35, in <module>
    key = acceptKey()
  File "/Users/mahika/SE PROJ/AES-Enc/caseStudy.py", line 7, in acceptKey
    raise Exception("Key length not 16")
Exception: Key length not 16
(base) mahika@Mahikas-MacBook-Air AES-Enc %
```

- Develop at least 3 mutants of the functioning code and test all 4 code bases using the test case from the first problem statement

Mutants:

(i) Giving 17 characteristics as password instead of 16:

```
PS D:\5th sem\SE\Case study-4> python -u "d:\5th sem\SE\Case study-4\original.py"
Enter key of 16 characters:asdfghjklmnbvcxzq
Traceback (most recent call last):
  File "d:\5th sem\SE\Case study-4\original.py", line 27, in <module>
    key = acceptKey()
  File "d:\5th sem\SE\Case study-4\original.py", line 6, in acceptKey
    raise Exception("Key length not 16")
Exception: Key length not 16
```

(ii) Input text is null:

```
PS D:\5th sem\SE\Case study-4> python -u "d:\5th sem\SE\Case study-4\original.py"
Enter key of 16 characters:asdfghjklmnbvcxz
Encrypted text: b''
Decrypted text:
PS D:\5th sem\SE\Case study-4>
```

(iii) Starting argument without an encoding

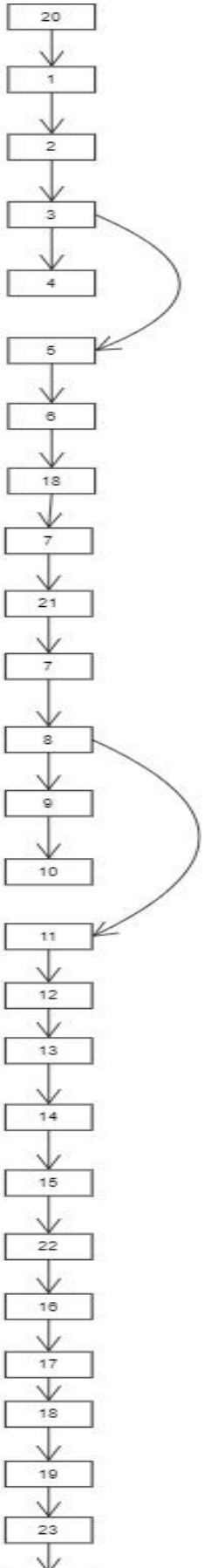
```
PS D:\5th sem\SE\Case study-4> python -u "d:\5th sem\SE\Case study-4\original.py"
Enter key of 16 characters:asdfghjklmnbvcxz
Traceback (most recent call last):
  File "d:\5th sem\SE\Case study-4\original.py", line 27, in <module>
    key = acceptKey()
  File "d:\5th sem\SE\Case study-4\original.py", line 7, in acceptKey
    key = bytes(key_string)
TypeError: string argument without an encoding
PS D:\5th sem\SE\Case study-4>
```

Static Testing

Static testing involves validating your code without any execution. Under this problem statement, you will be expected to analyze and calculate the cyclomatic complexity of your code.

- Using the unit you selected in the first problem statement as an example, develop the control flow graph of your problem statement.

```
1) def acceptKey():
2)     key_string = input("Enter key of 16 characters:")
3)     if len(key_string)!=16:
4)             raise Exception("Key length not 16")
5)     key = bytes(key_string, 'utf-8')
6)     return key

7) def encryptText(key,data):
8)     if data == None:
9)             raise Exception("No data entered")
10)            exit()
11)    cipher = AES.new(key,AES.MODE_EAX)
12)    nonce = cipher.nonce
13)    data = "hi this needs to be encrypted".encode()
14)    ciphertext = cipher.encrypt(data)
15)    return ciphertext,nonce

16) def decrypt(key,ciphertext,nonce):
17)    cipher = AES.new(key,AES.MODE_EAX,nonce=nonce)
18)    plaintext = cipher.decrypt(ciphertext).decode()
19)    return plaintext

       data = None
20) key = acceptKey()
21) ct,nonce = encryptText(key,data)
22) pt = decrypt(key,ct,nonce)
23) print("Encrypted text:", ct)
24) print("Decrypted text:", pt)
```



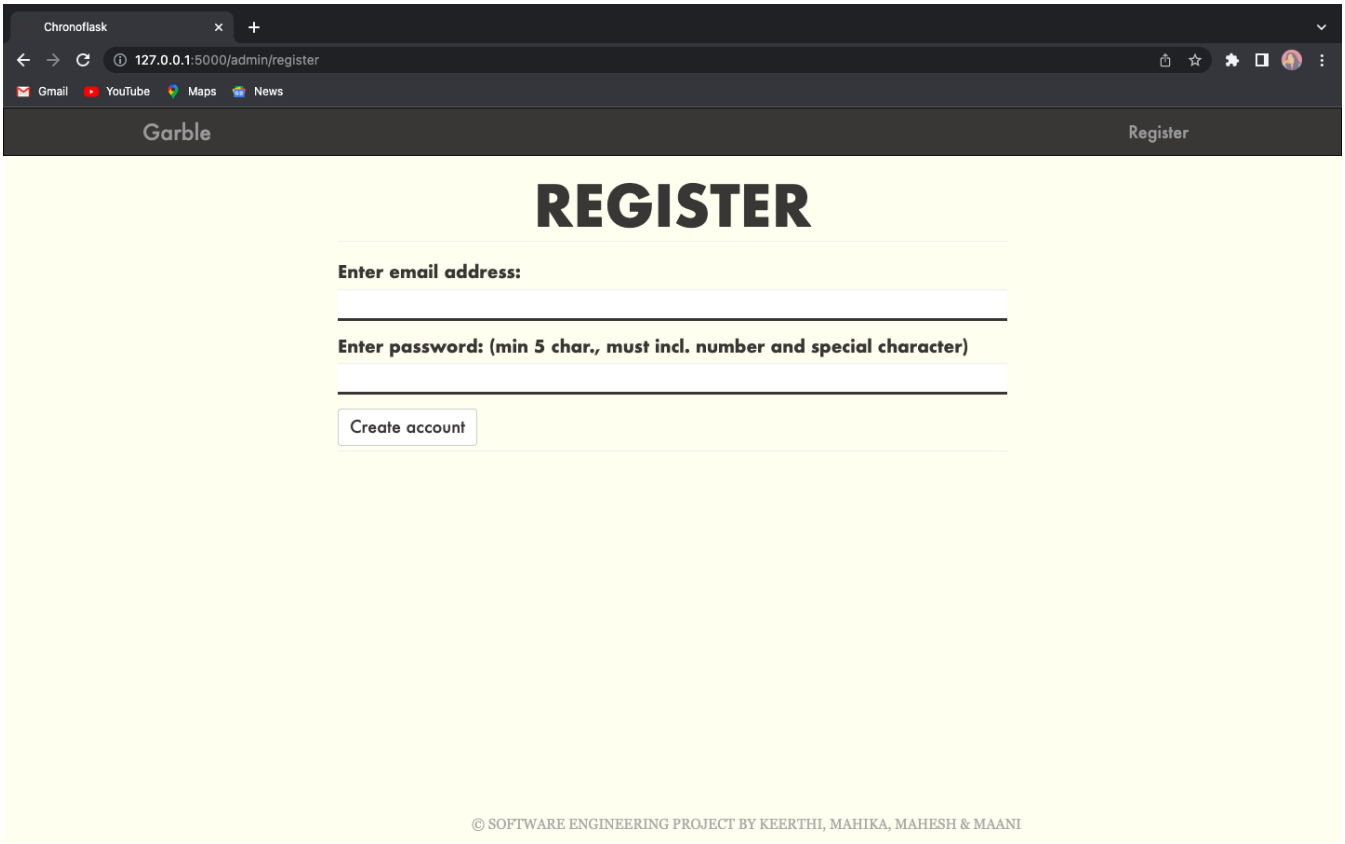
- Using the Control flow graph, calculate the cyclomatic complexity of your code.

Cyclomatic complexity = N - M + 2P
{where, N=no of Nodes, M=No f edges, P=No of exit points} Cyclomatic complexity = 6(N = 24, M = 24, P = 3)

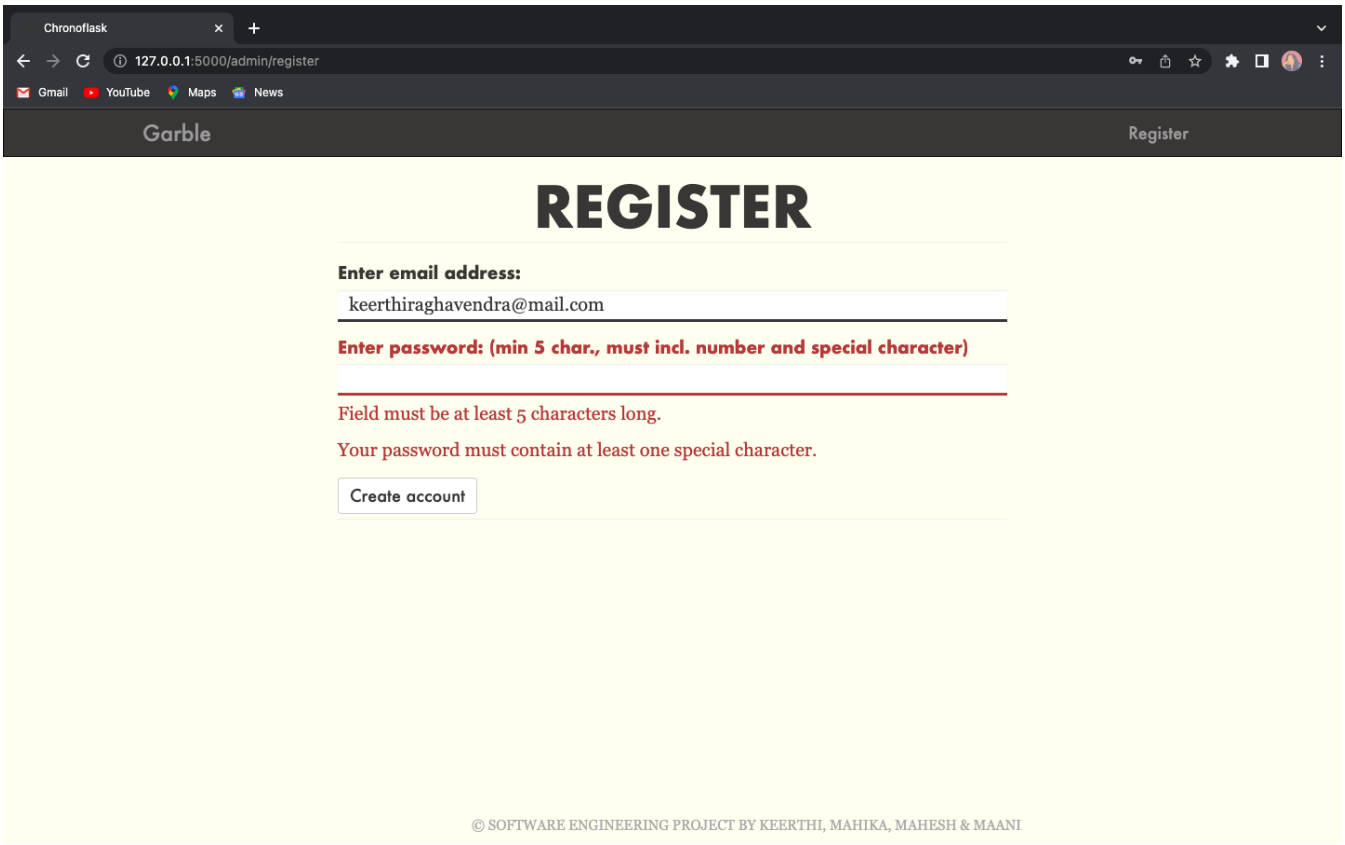
- Using the cyclomatic complexity as an indicator, Ideate and code your unit again to reduce complexity

Cyclomatic complexity = 21 - 21 + 2(2) = 4
{After removing if statement in 'encrypttext' function}

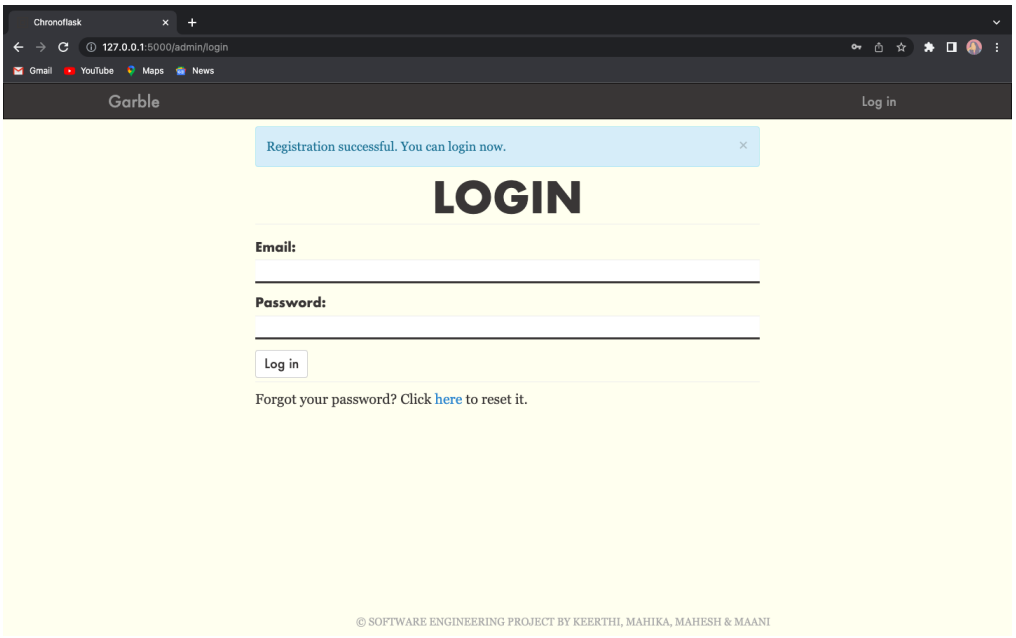
Similarly for all the functions the test cases are ss follows:



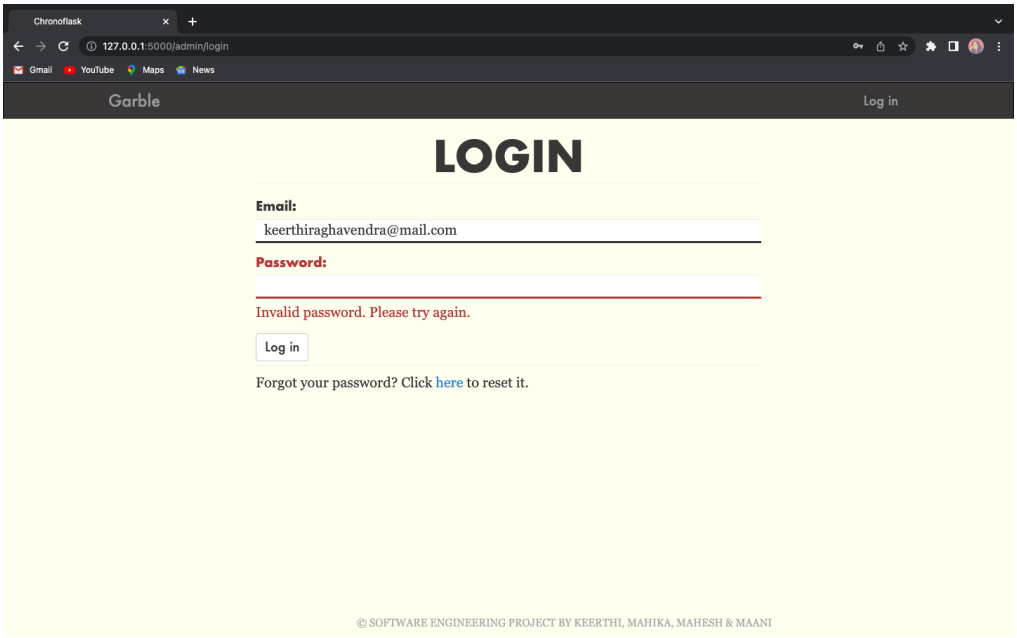
The first step in our project is to register a user. The database presently has no users and we must register a new user to begin using our application.



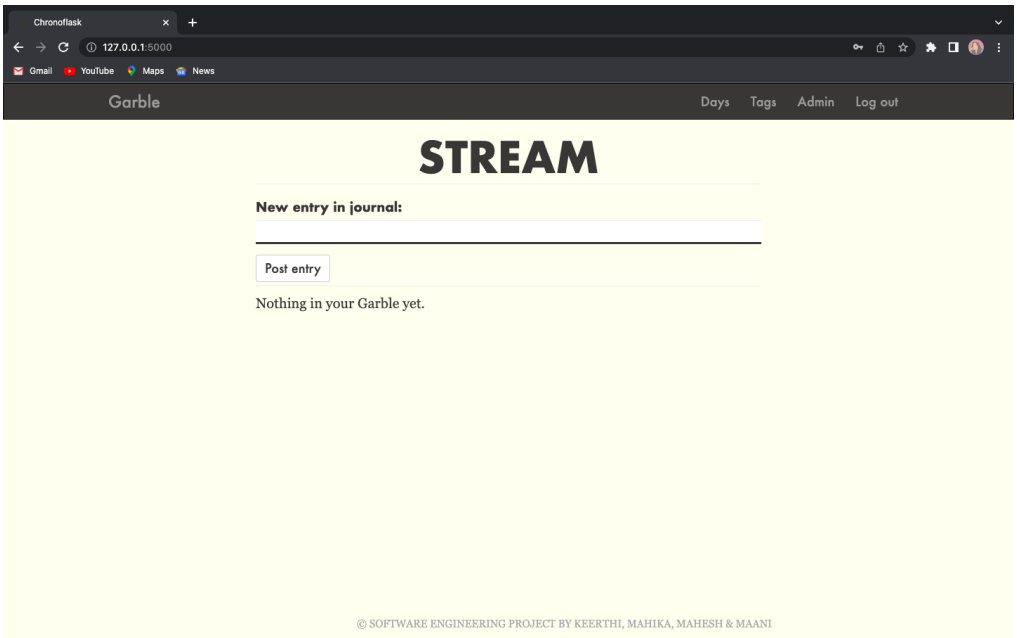
The boundary condition for a password that must be set with the given conditions is handled gracefully.



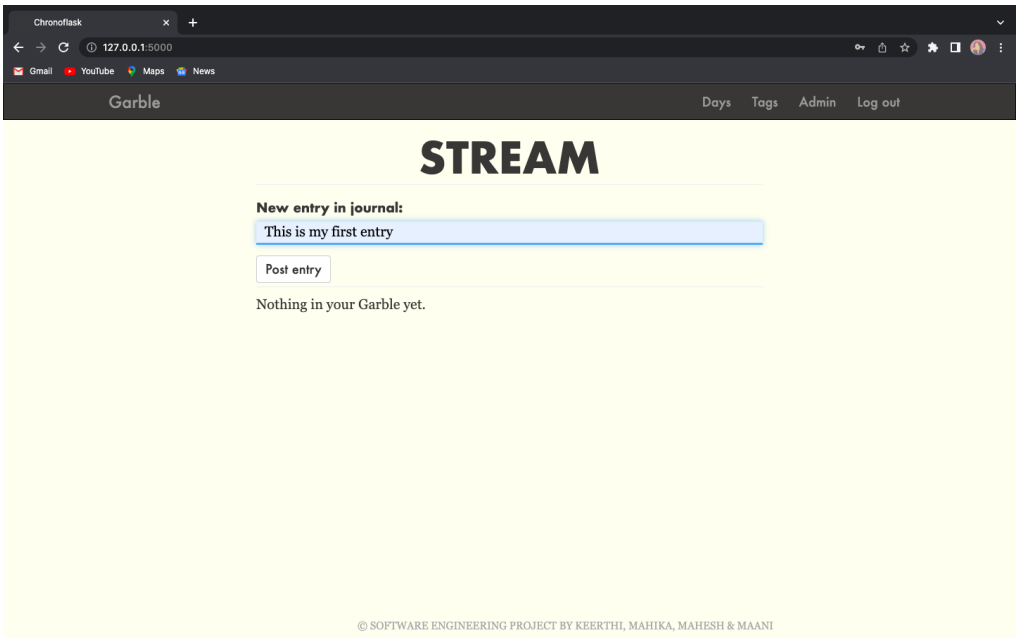
You get a flash message when all the details are entered correctly and the registration is successful.



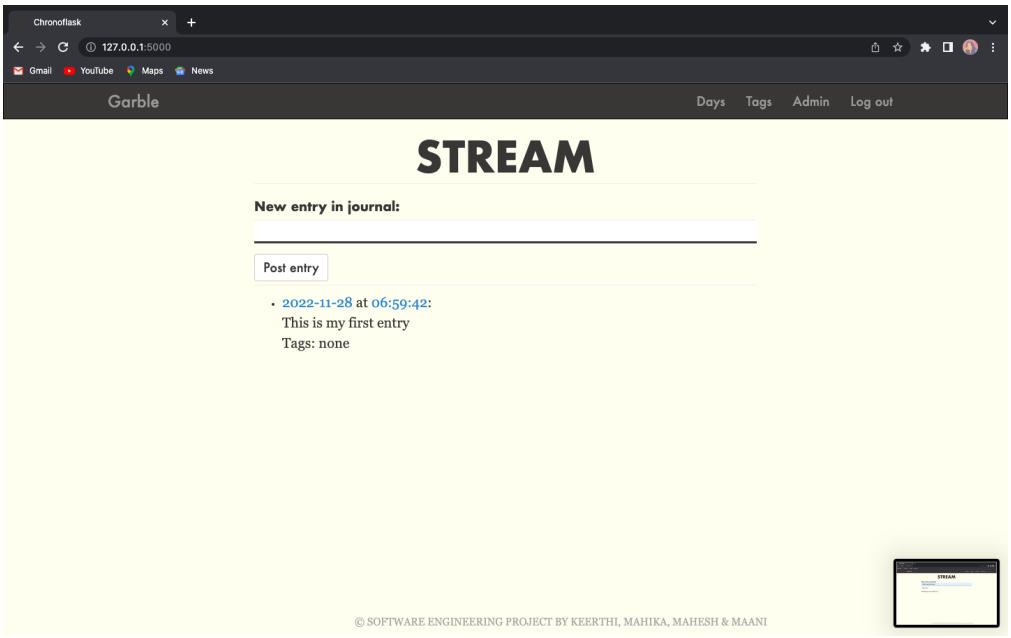
The application now requires you to login as the user you just registered. The boundary condition for invalid username and password is handled.



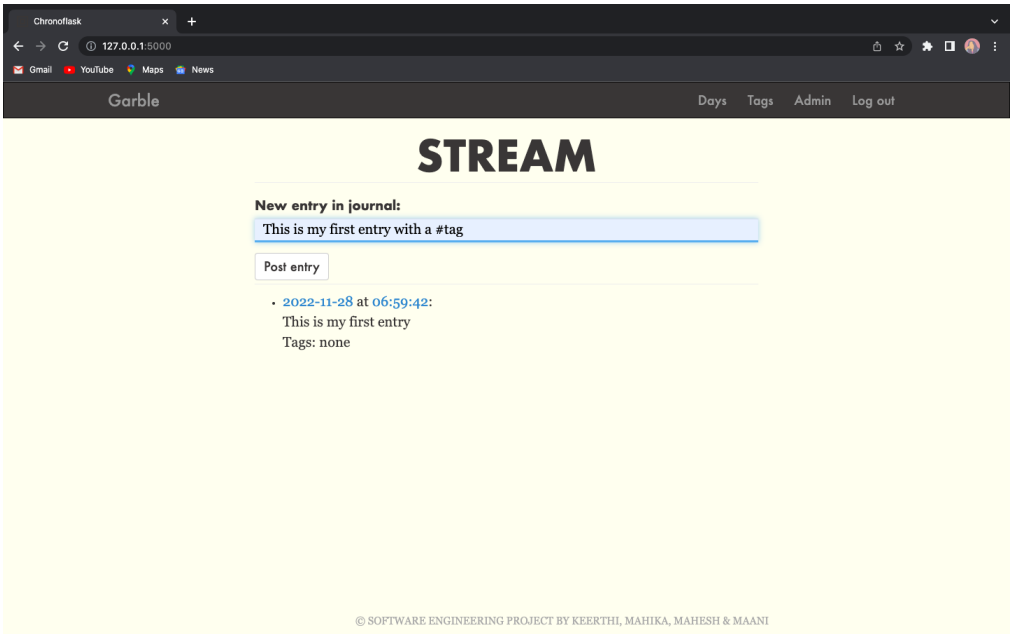
On successful login, you see the homepage which consists of the journal entry and the stream of journal entries. The different options can be seen in the homepage for easy navigation.



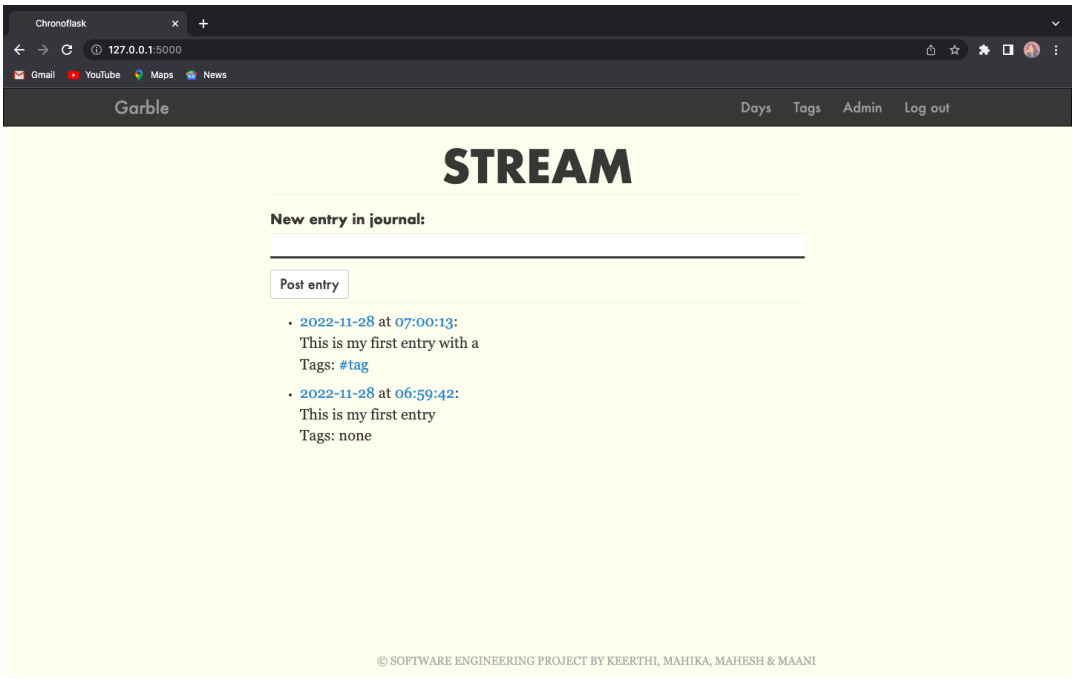
A new entry is typed in.



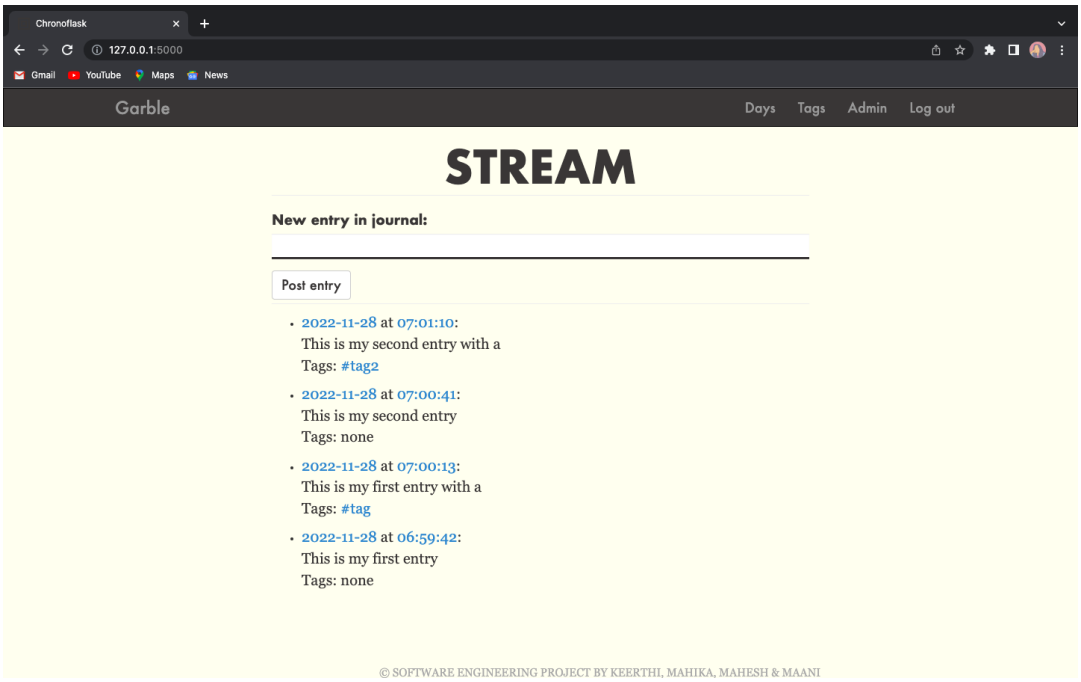
Once the post entry button is hit, a new entry is made in the journal with the given date and time.



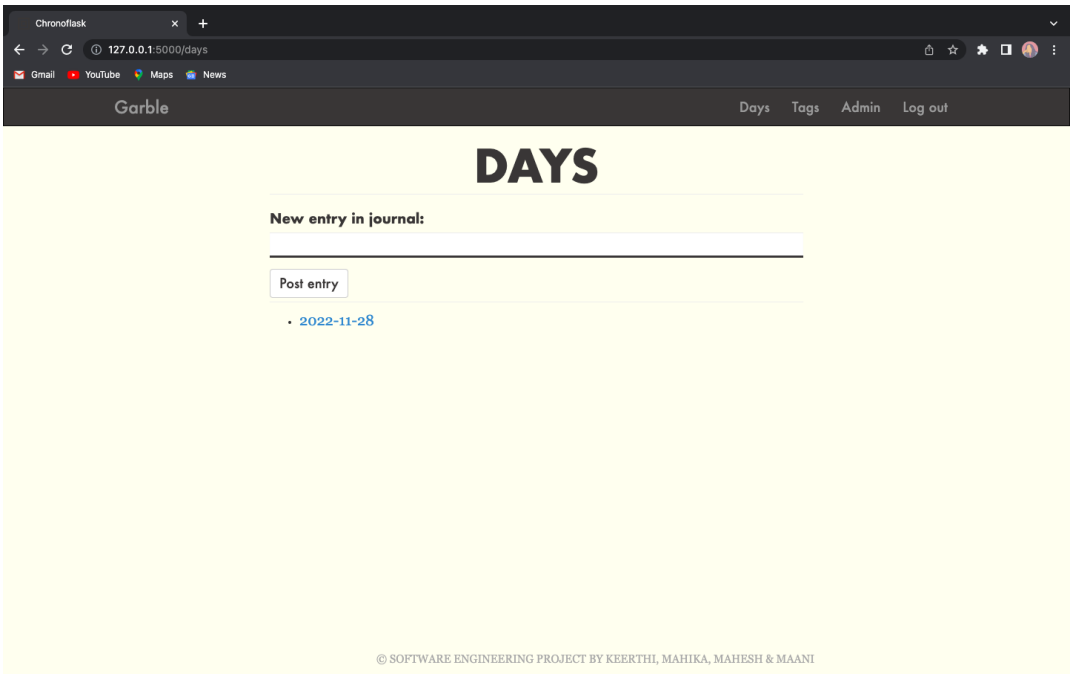
We can also have entries with “#” tags. These entries can be later filtered based on the tags!



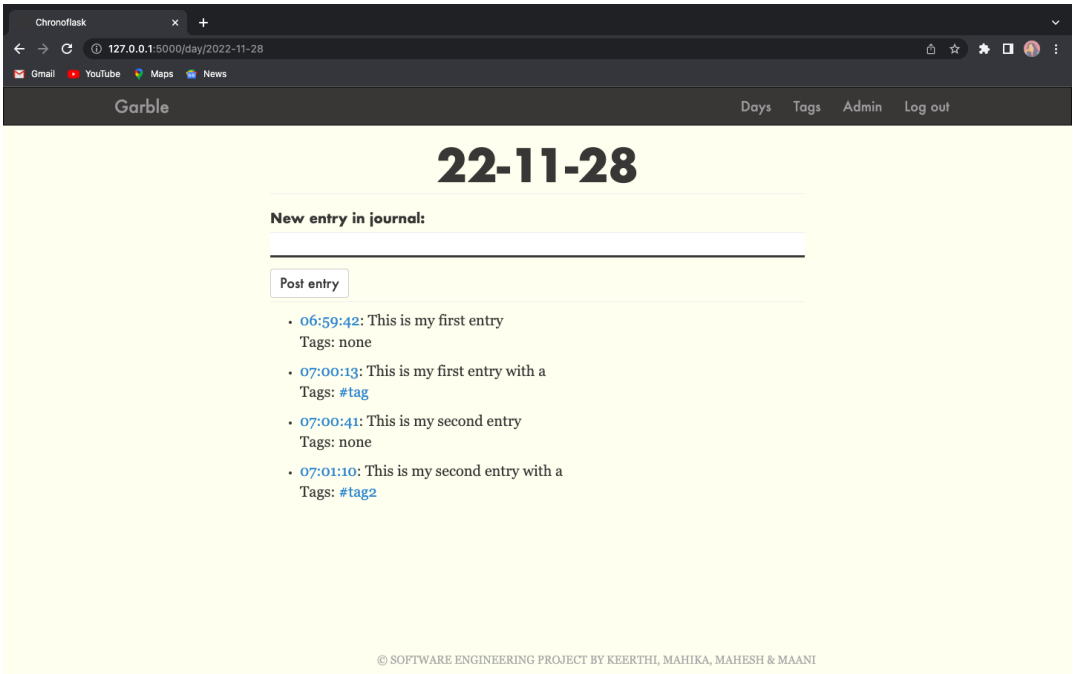
On posting this entry, we can see that the tag is classified under “Tags:”.



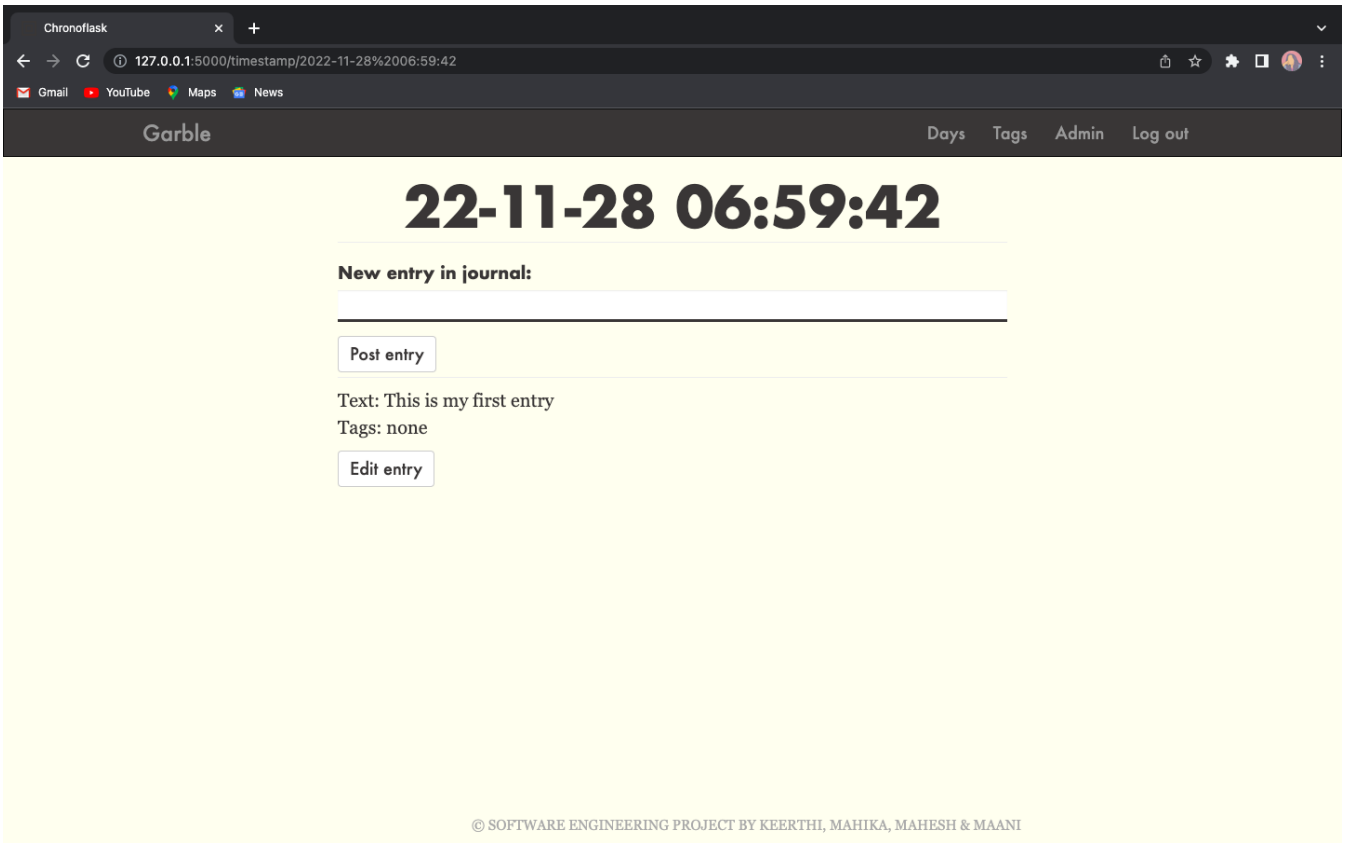
We now populate the journal with more entries.



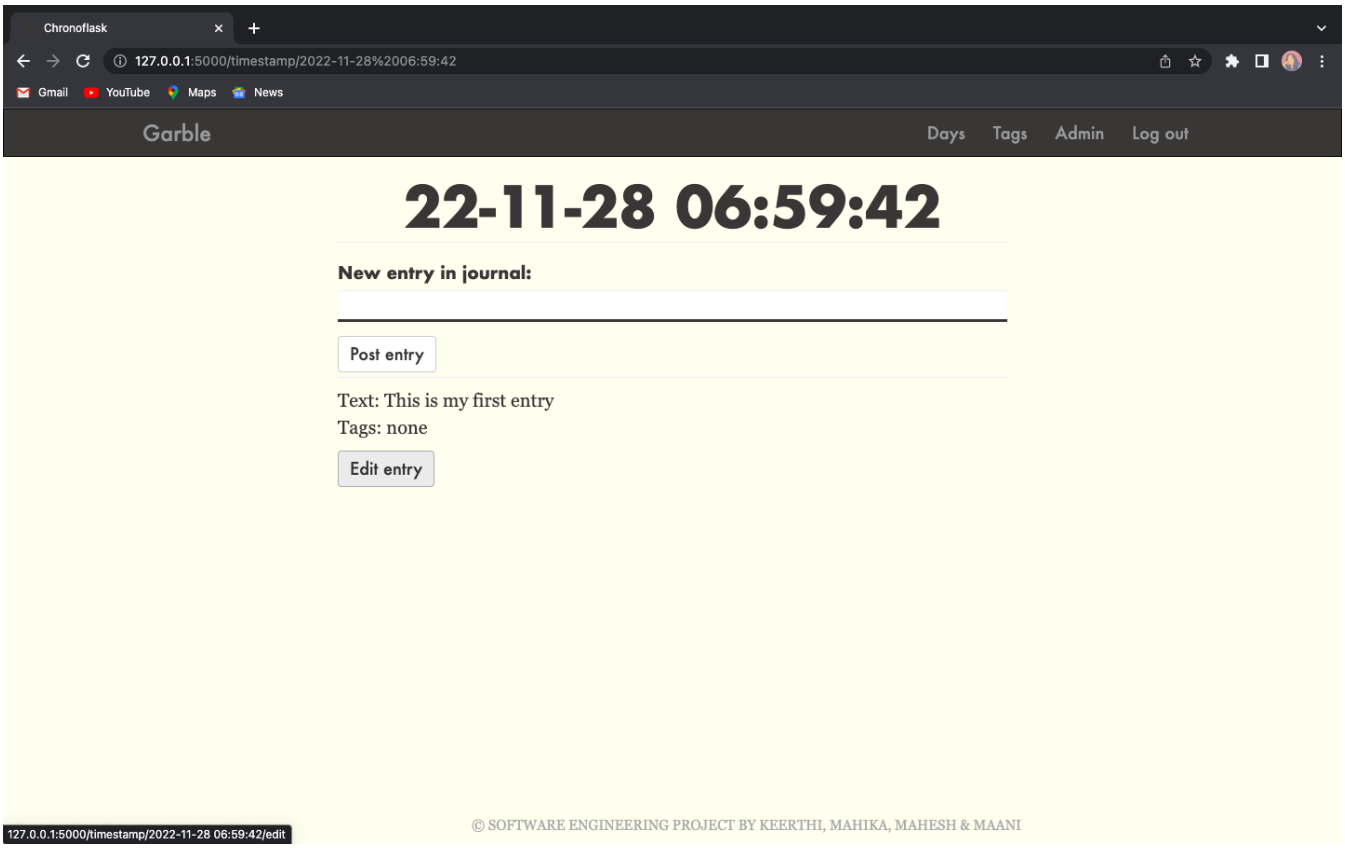
We now look at the "Days" implementation which will consist of a list of all entries made on a particular date.



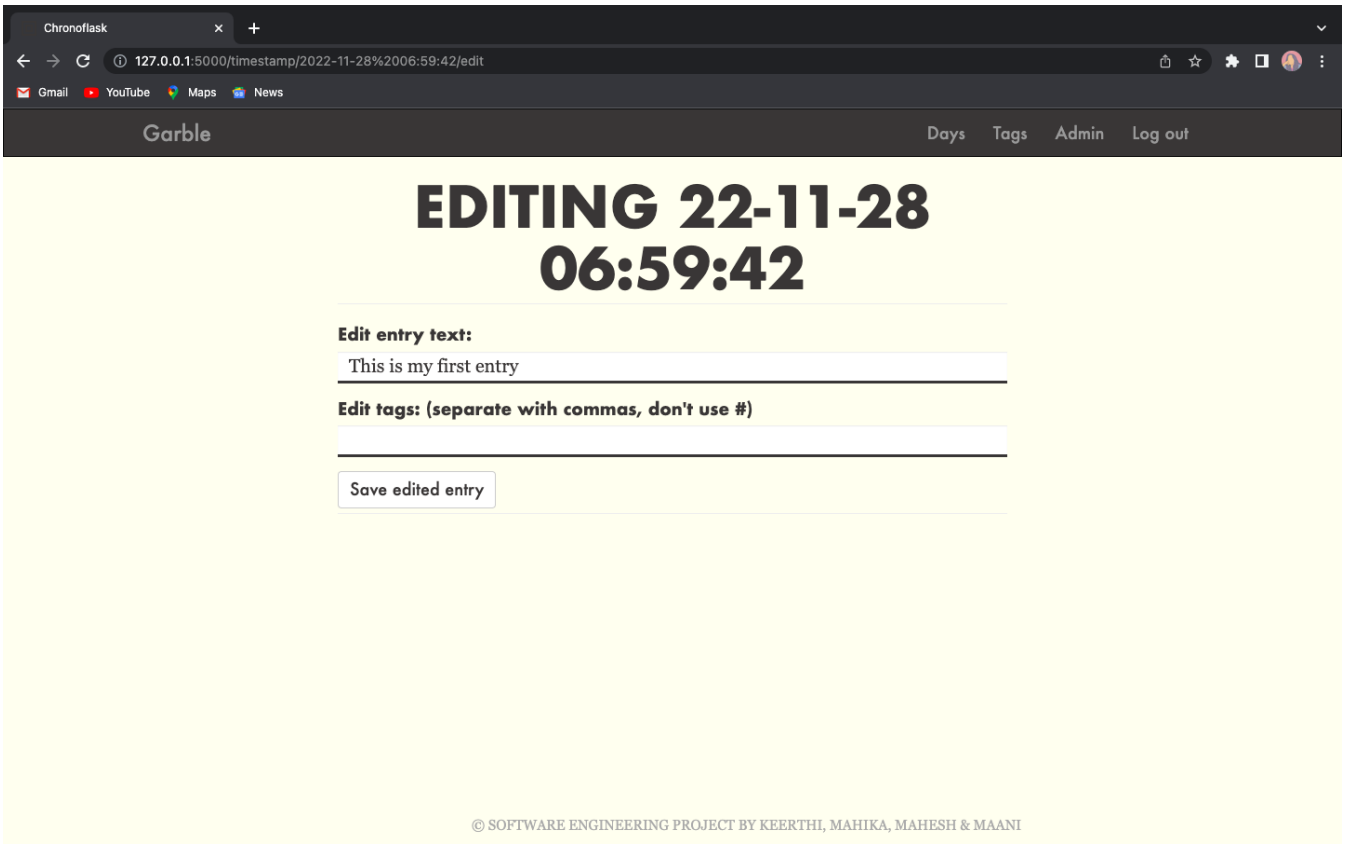
When a particular date is selected, we can see all the entries made on that date with the tags.



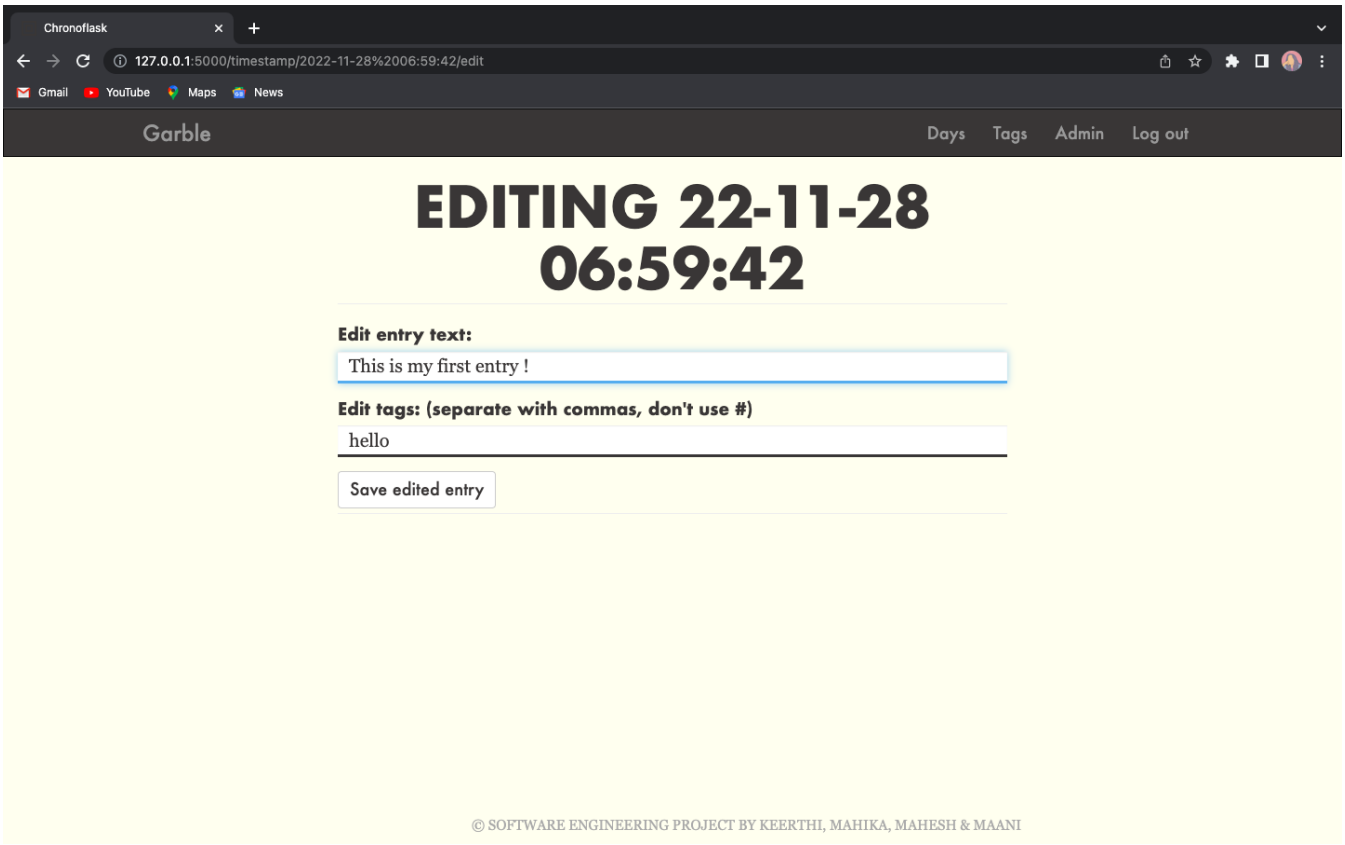
If we now select on the particular time an entry was made on a particular date we see that the entry is viewed separately. Now, we have two options, you can either post a new entry or edit the entry taht you are currently viewing.



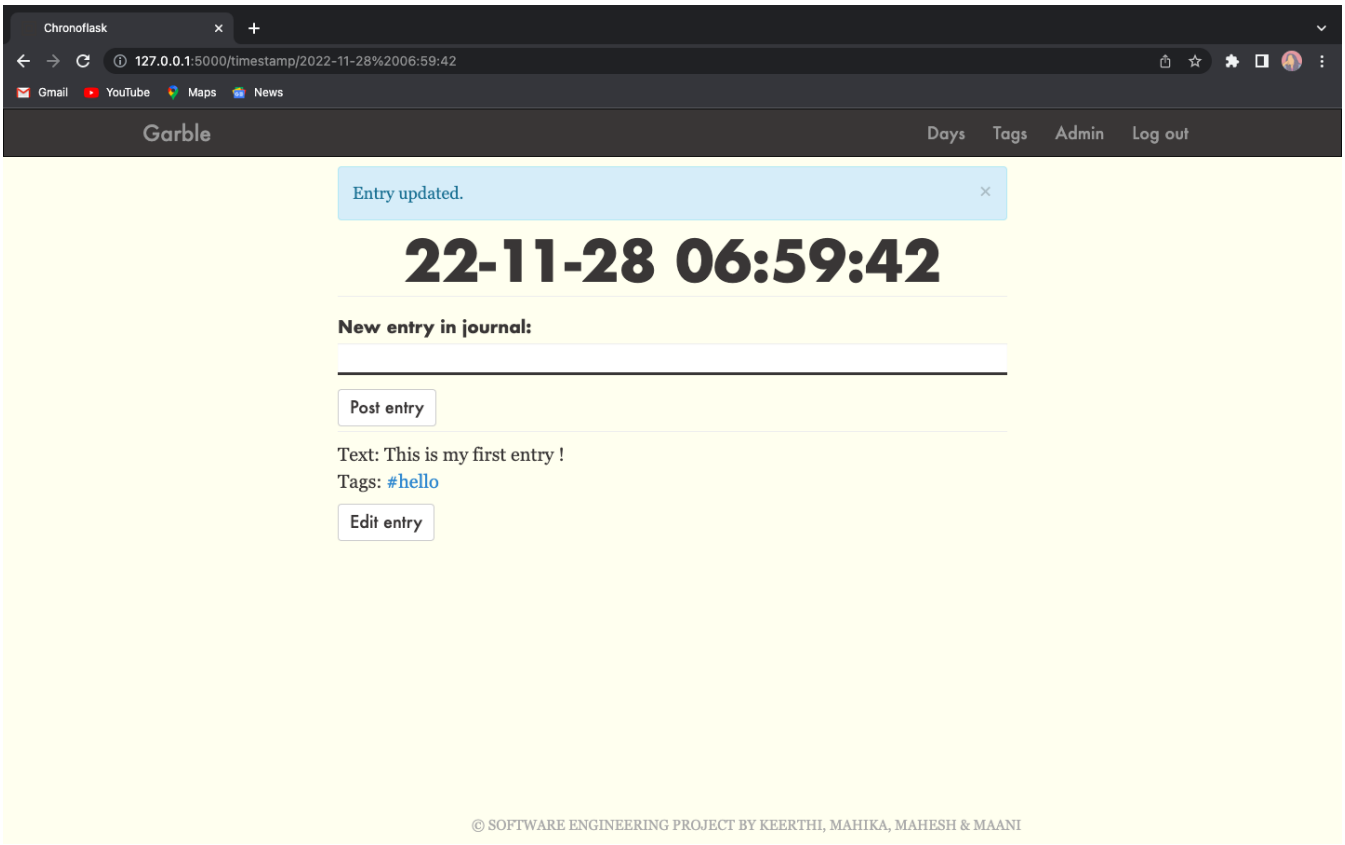
We click on the edit entry.



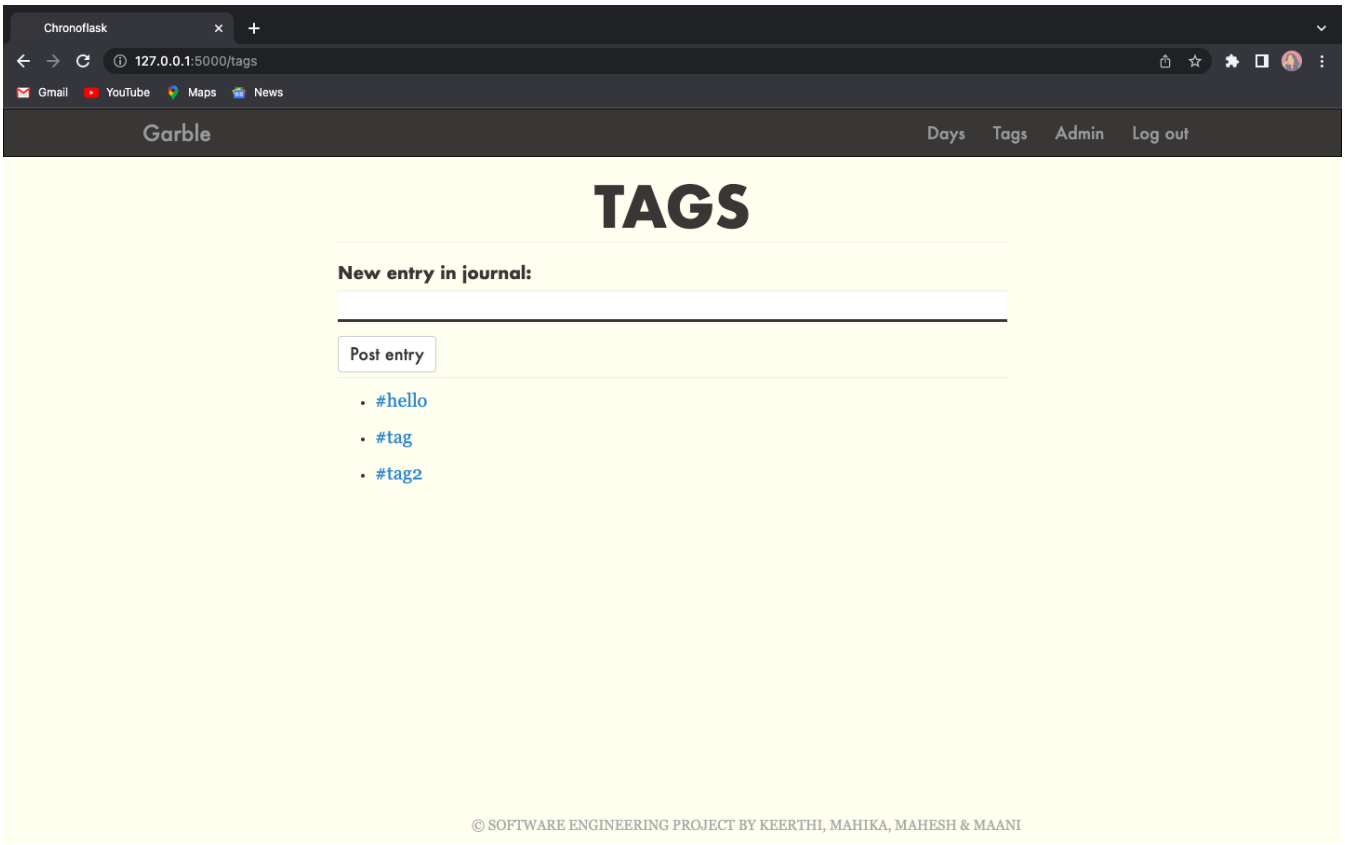
In the editing page we can either edit the text matter of the entry or the tags.



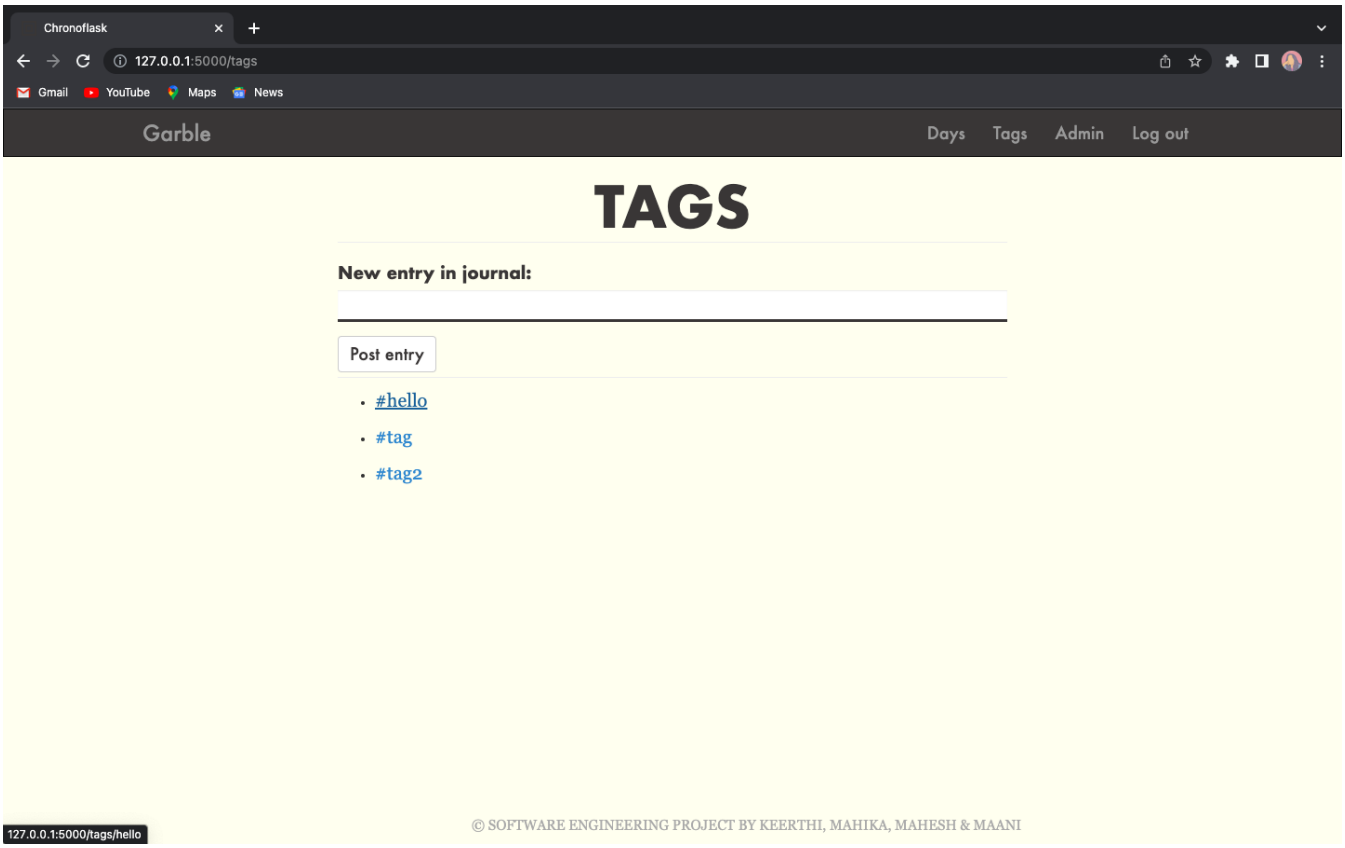
We have added a "!" to the entry text and added a "hello" tag.



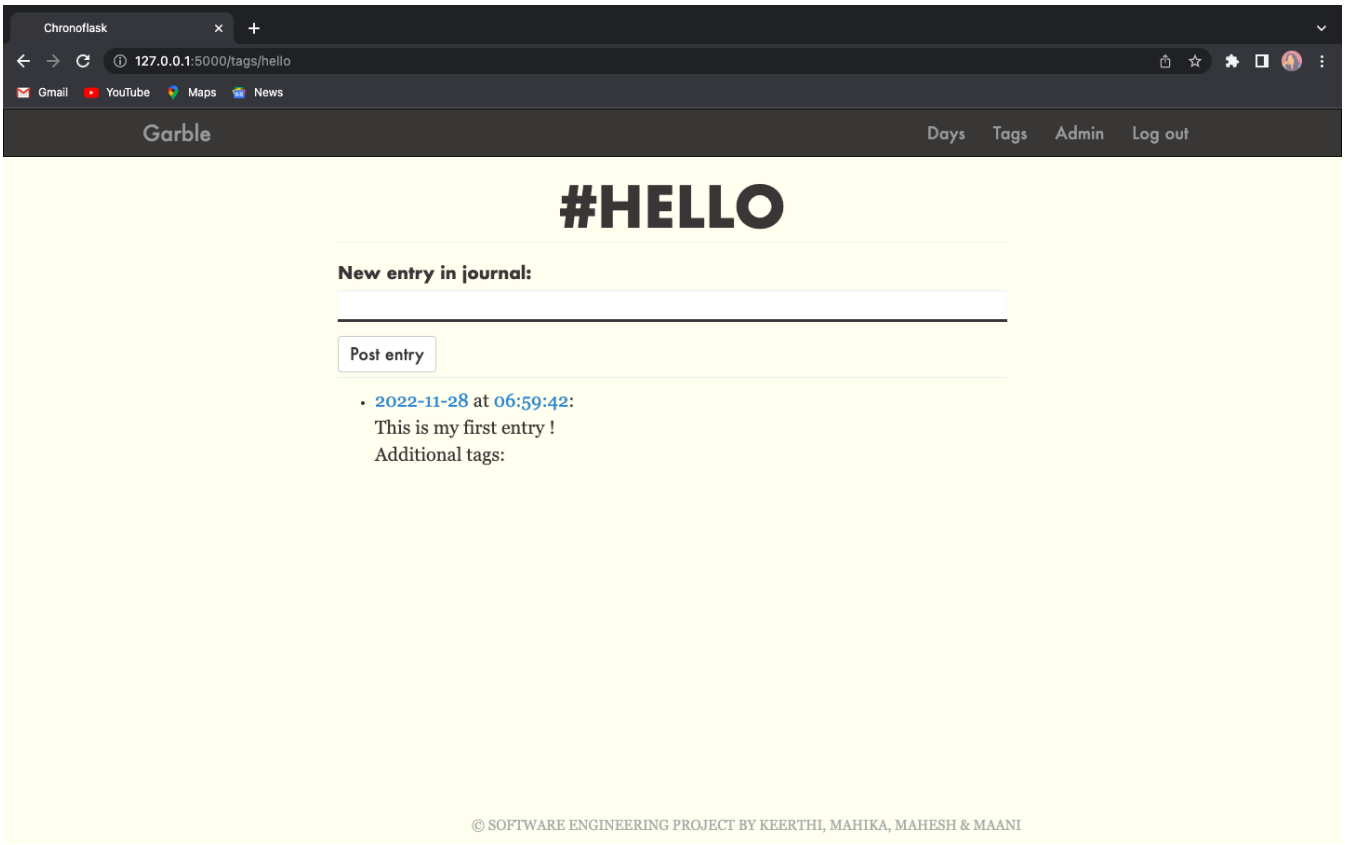
We see that the entry is successfully updated and we can see the changes made reflected here.



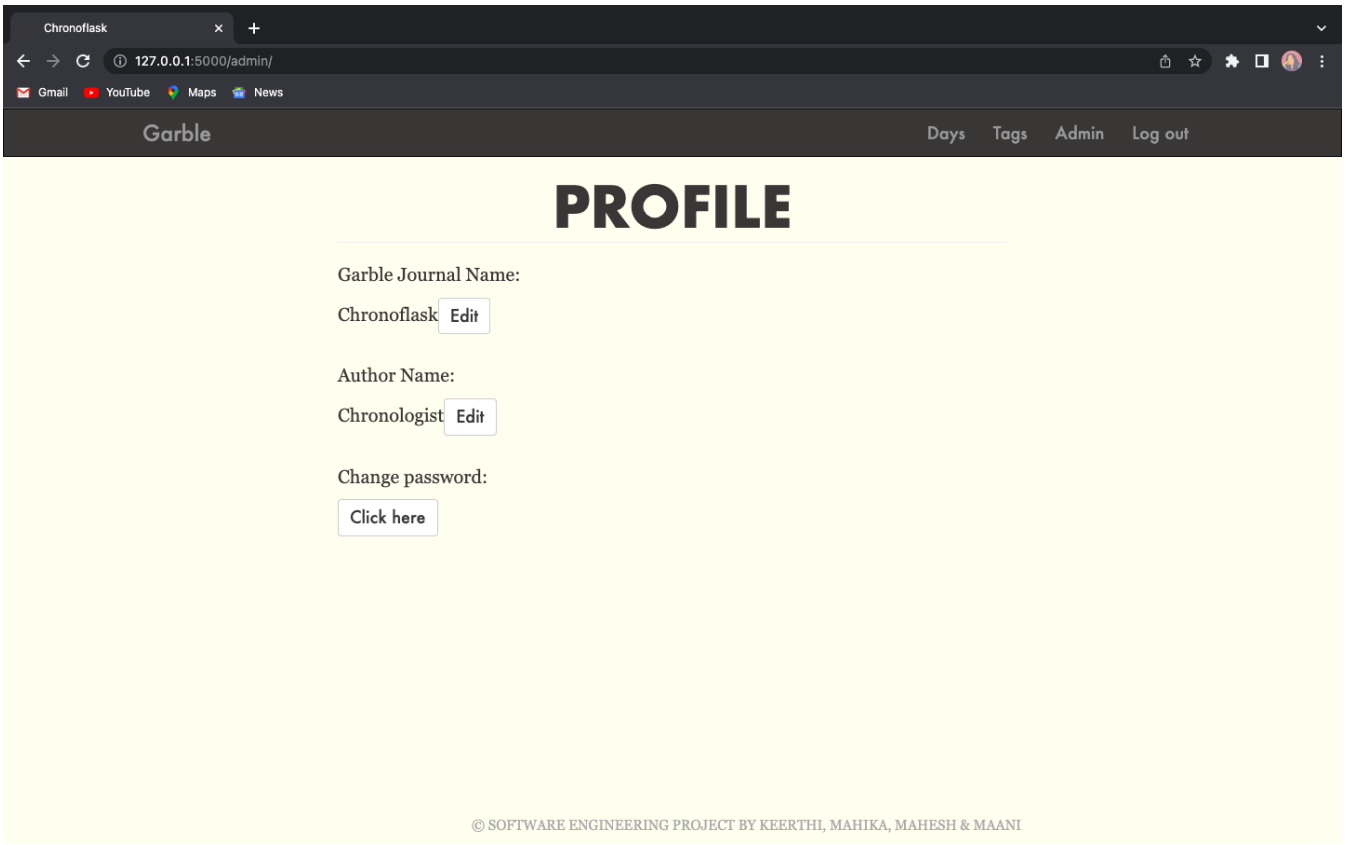
Now, when we look into the "Tags" page we can see all the tags used in all the entries of the journal.



On clicking a particular tag…



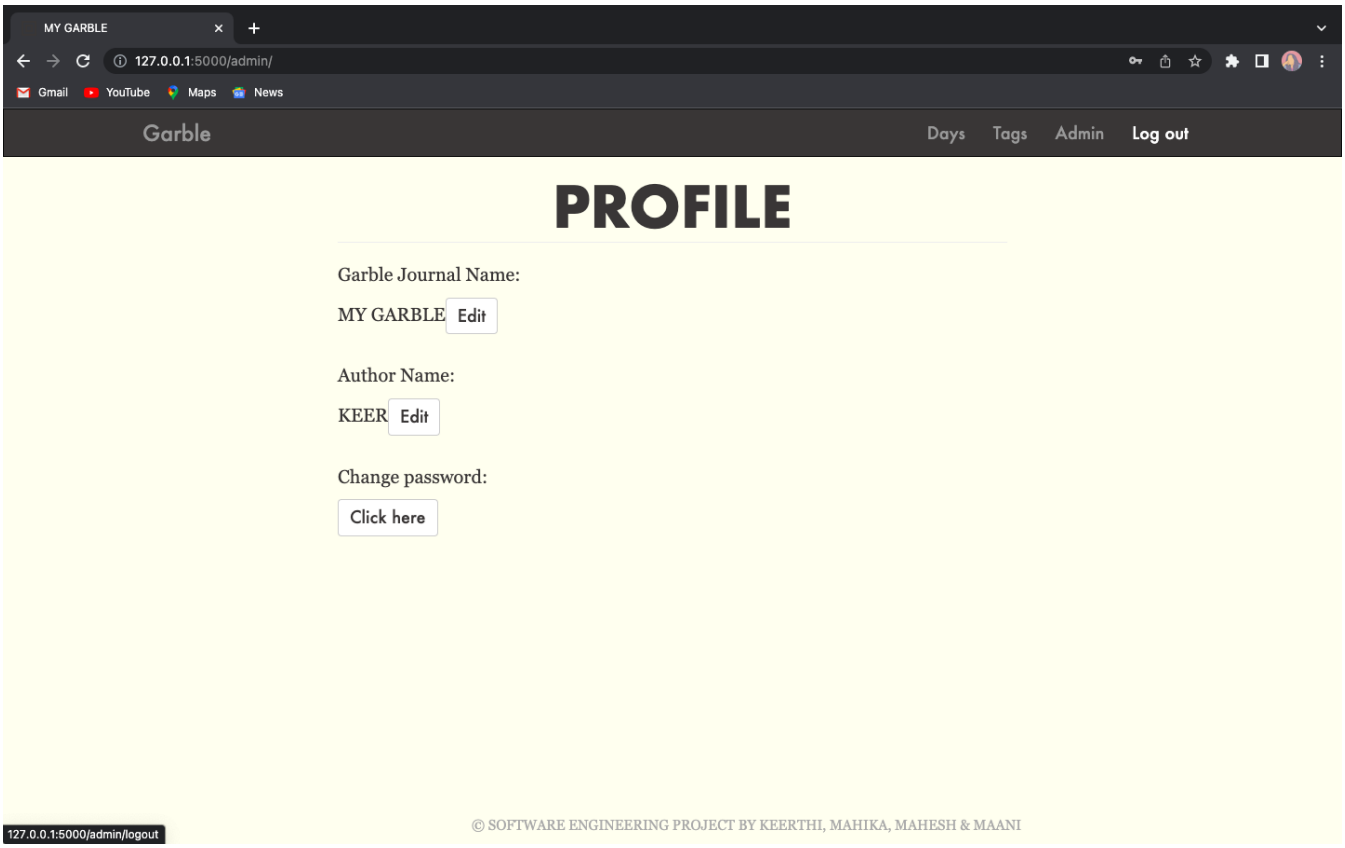
We can go to the entry/entries that contain the tag in its entry.



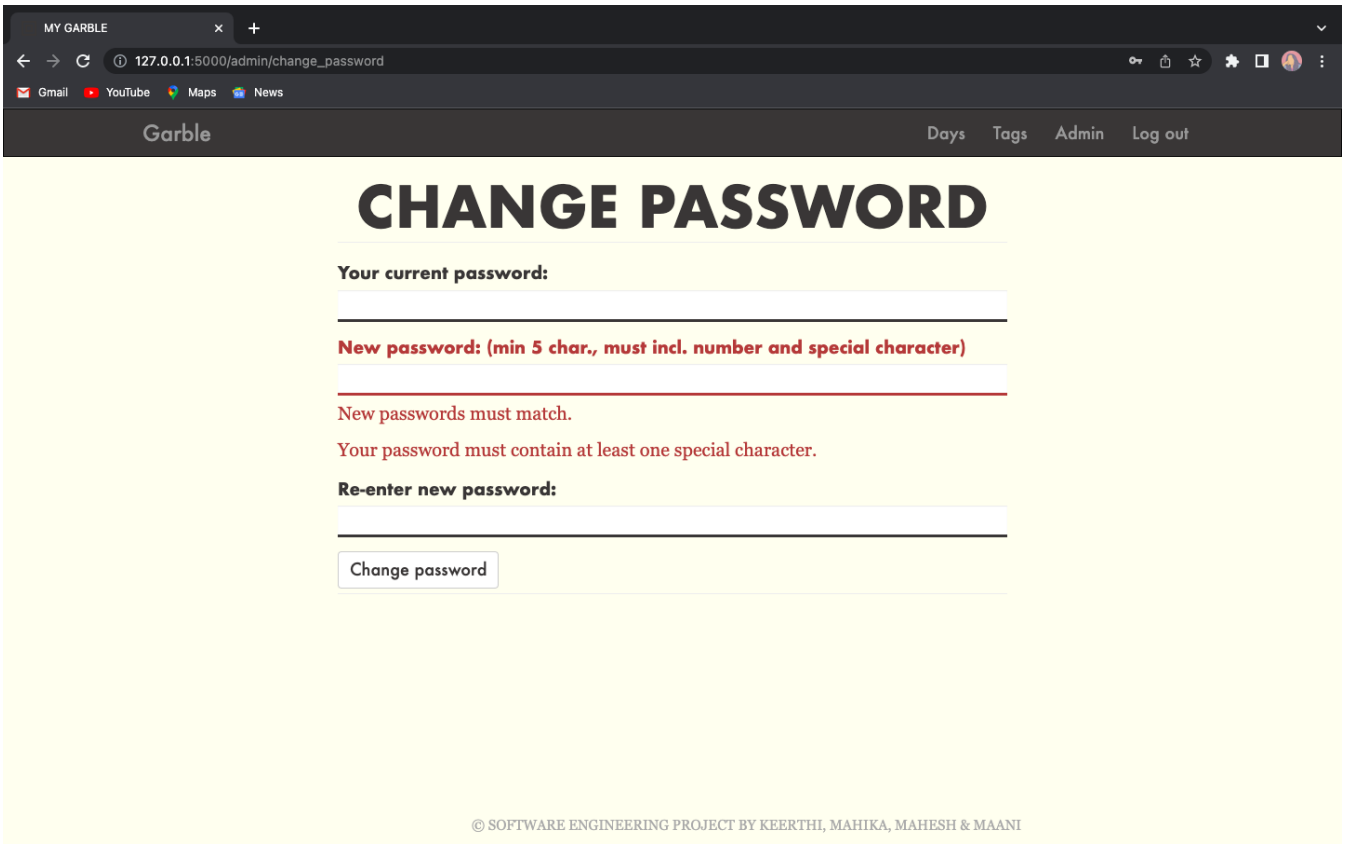
In the Profile/Admin page, we can change the name of the journal which will be reflected as the title of the site. We can also change the authors name of the journal to help personalize the feel of using this journal.

Chronoflask is the default term coined for this journal as it arranges all the entries in a chronological order and we have used flask to execute this.

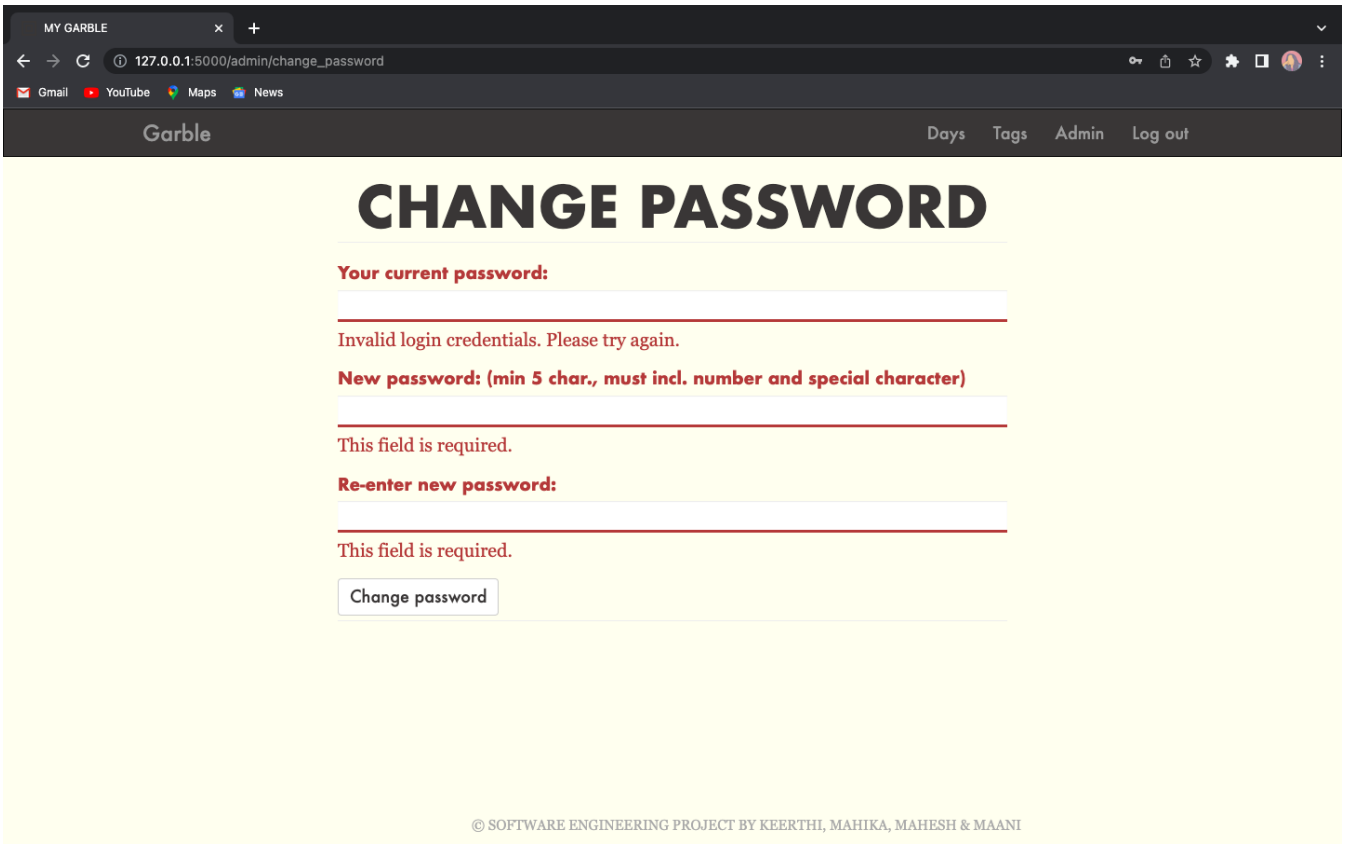
Chronologist is the author name as it's a person who does things in a chronological order! The default names can be changed within the application.



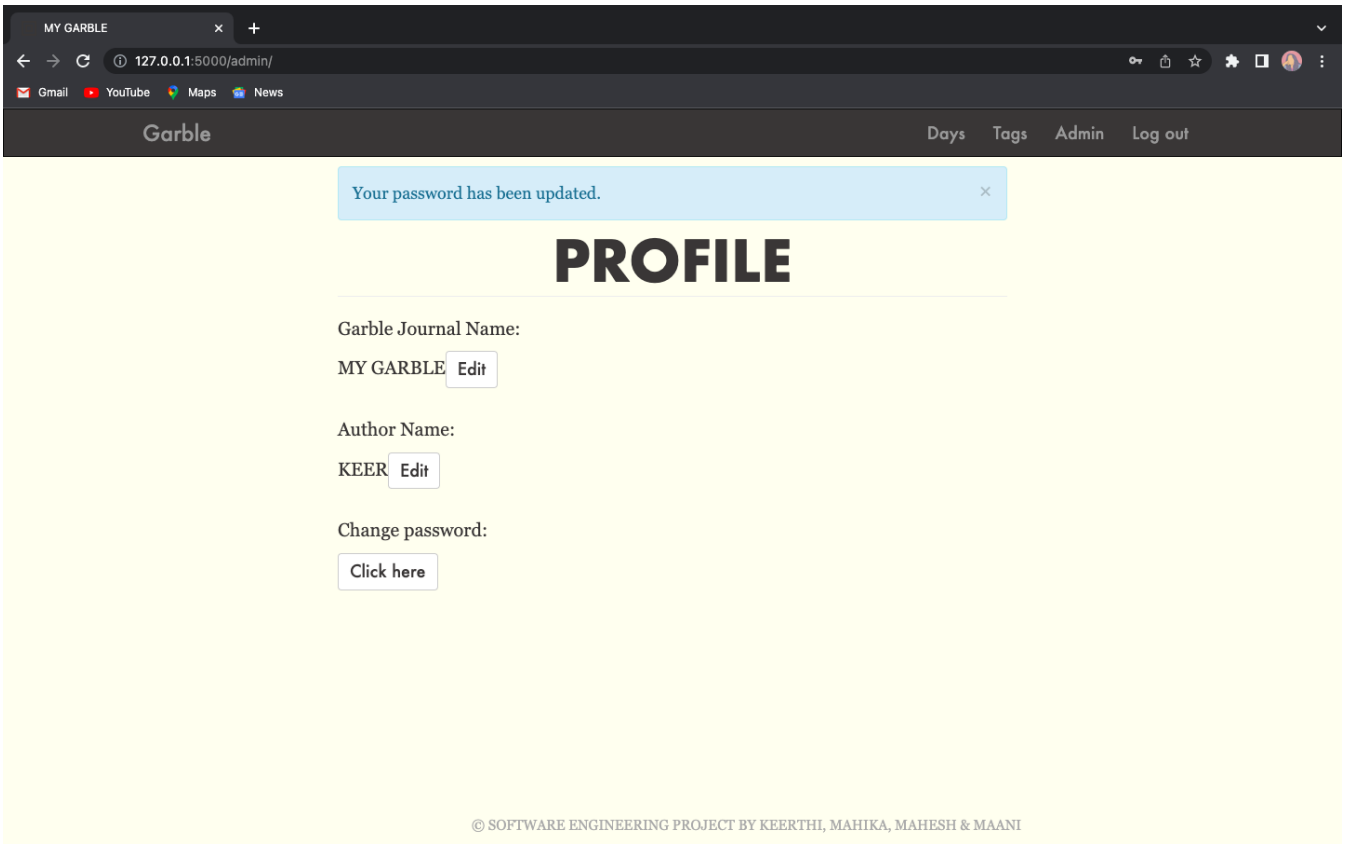
Here we can see that the Garble journal names as well as the author name has been successfully updated.



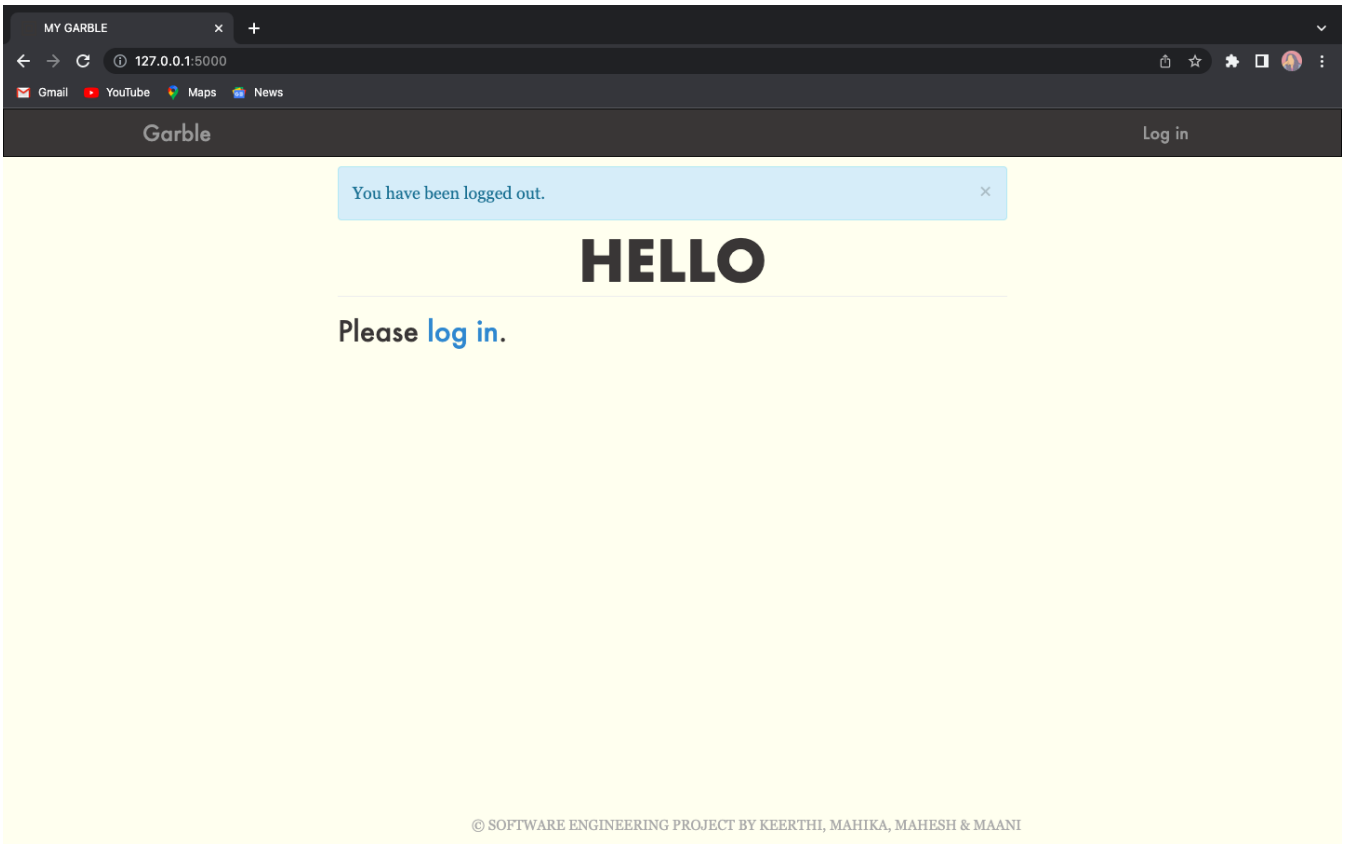
If you choose to change your password, as you can see all the boundary conditions has been handled successfully.



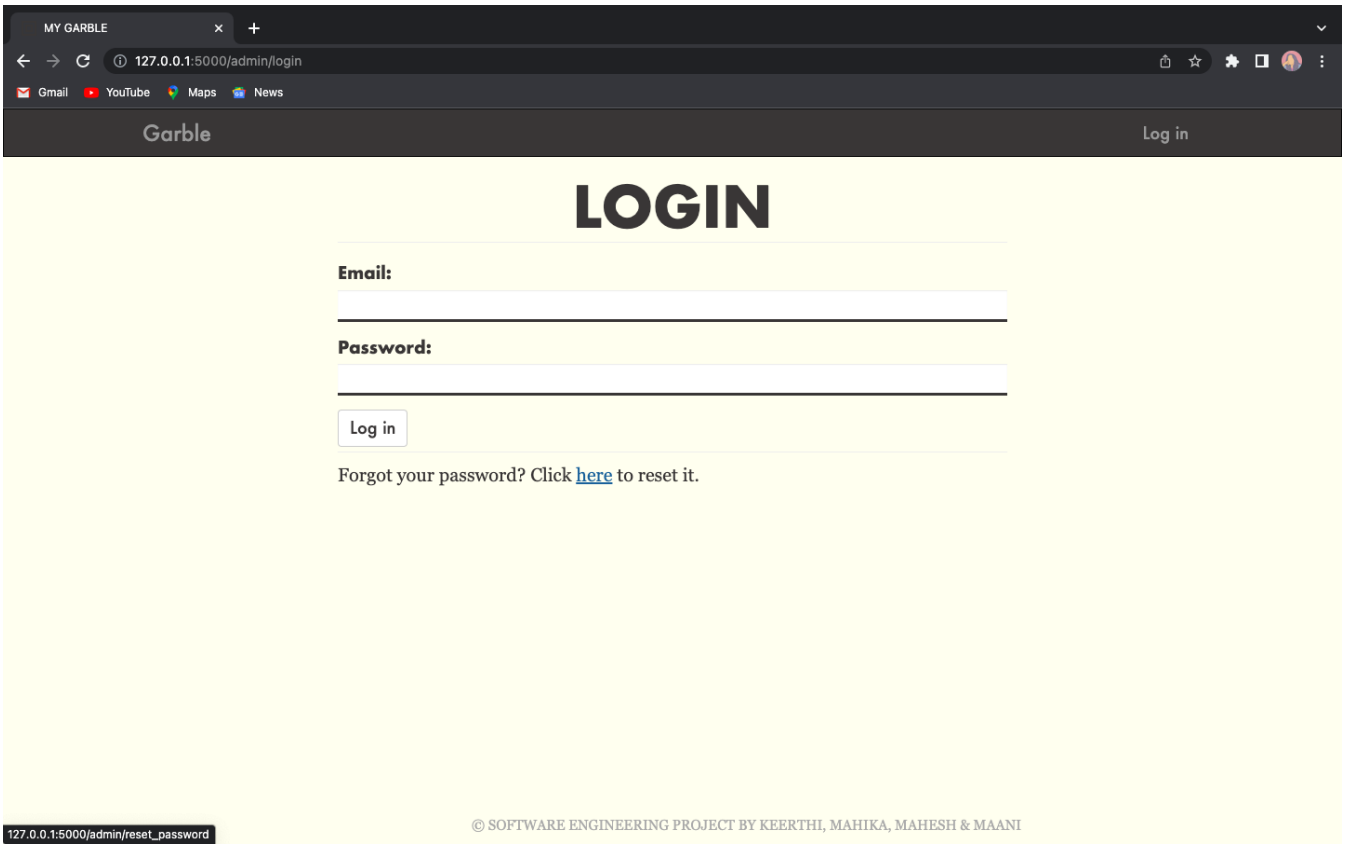
We can see that all boundary conditions have been implemented.



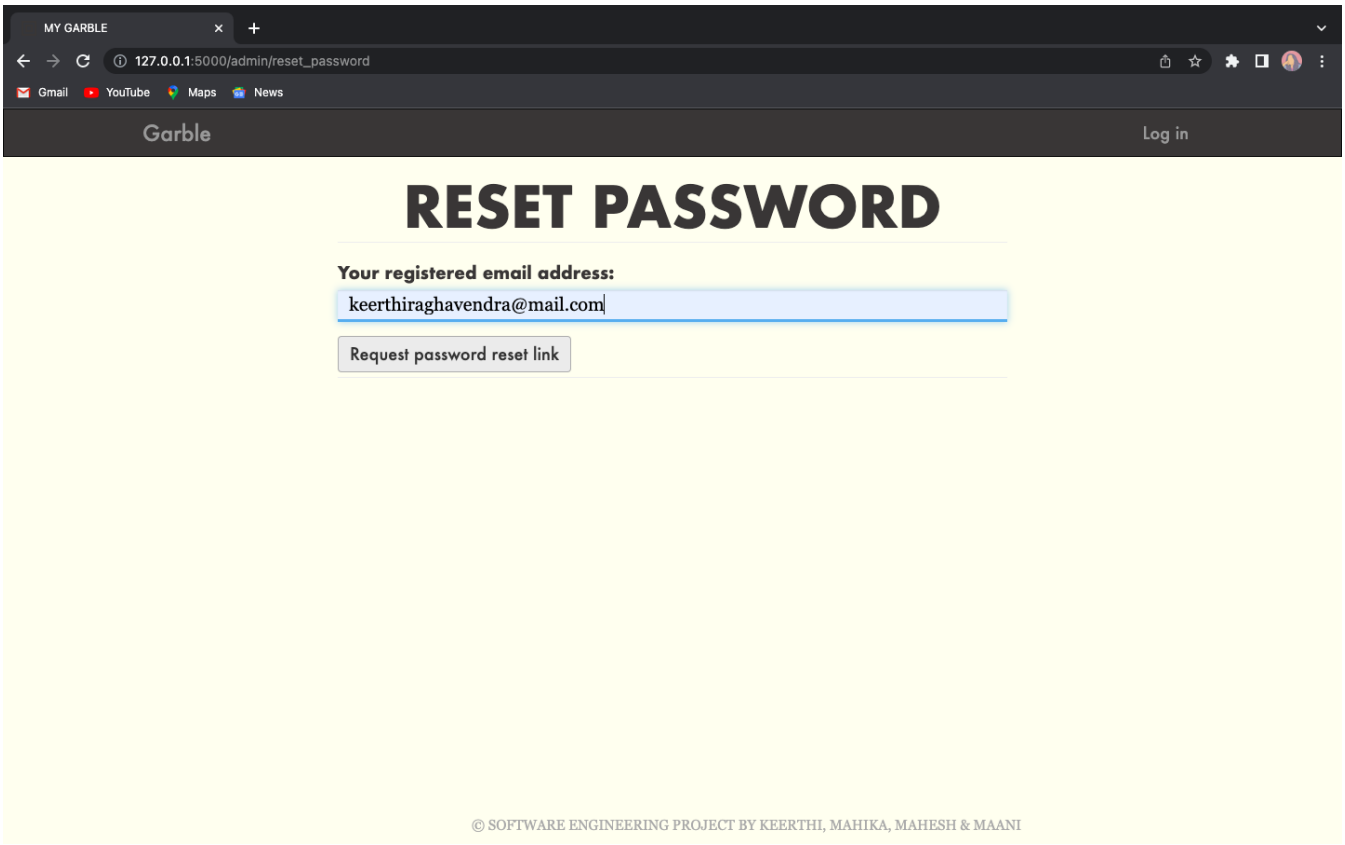
Once successful, we can see that the password has been updated successfully!



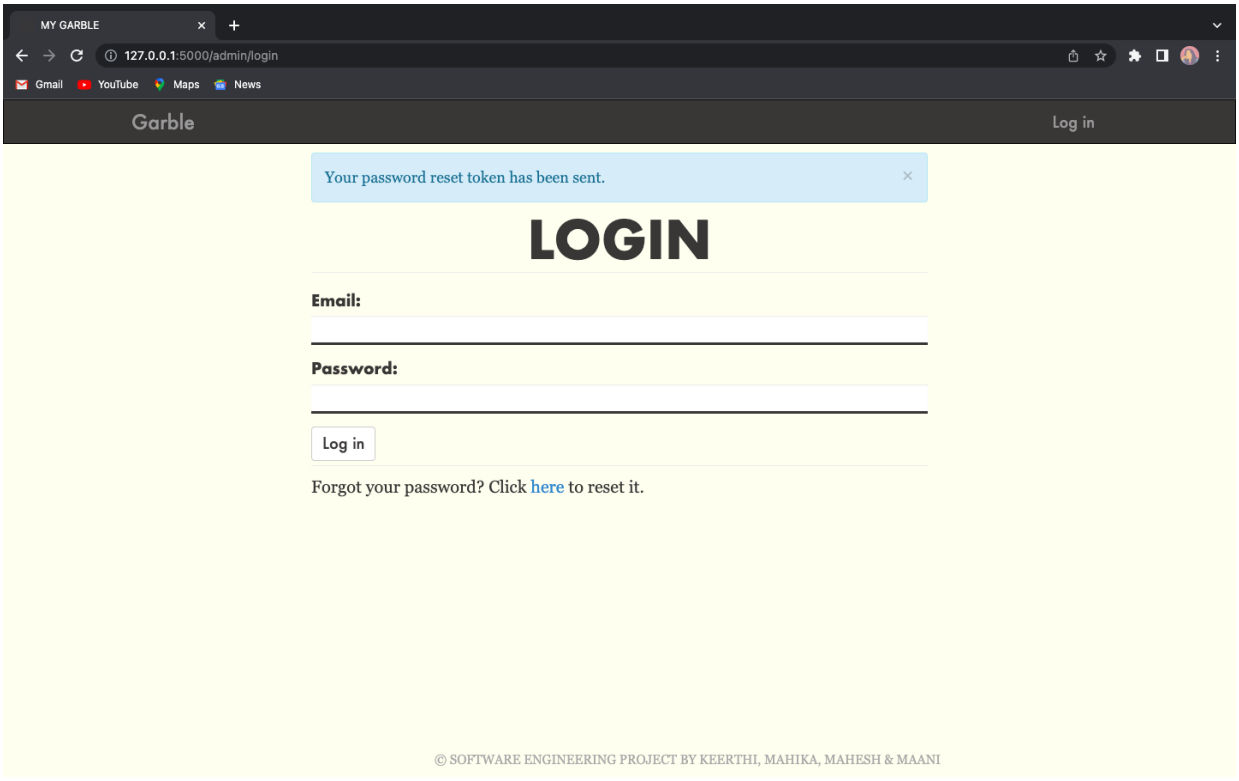
On the top right corner, we can choose to log out which will bring us back to the home page, with an option to login to continue using our application.



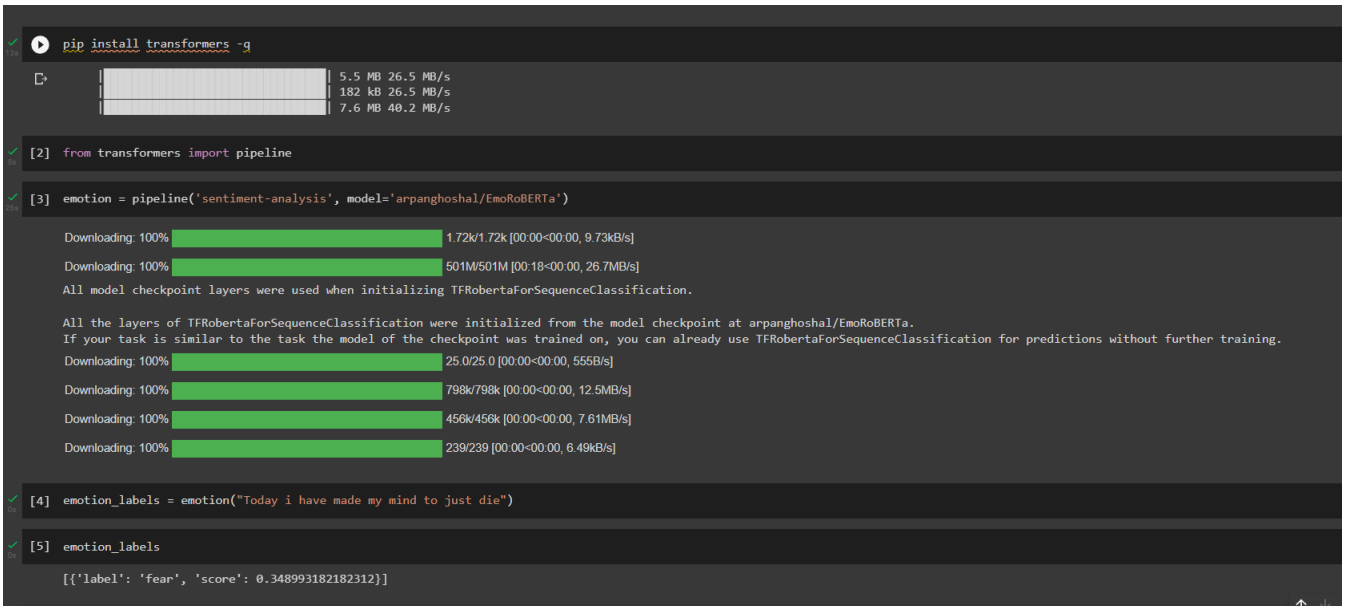
In case you have forgotten your password during login, you can choose to reset it based on the username that was initially given.



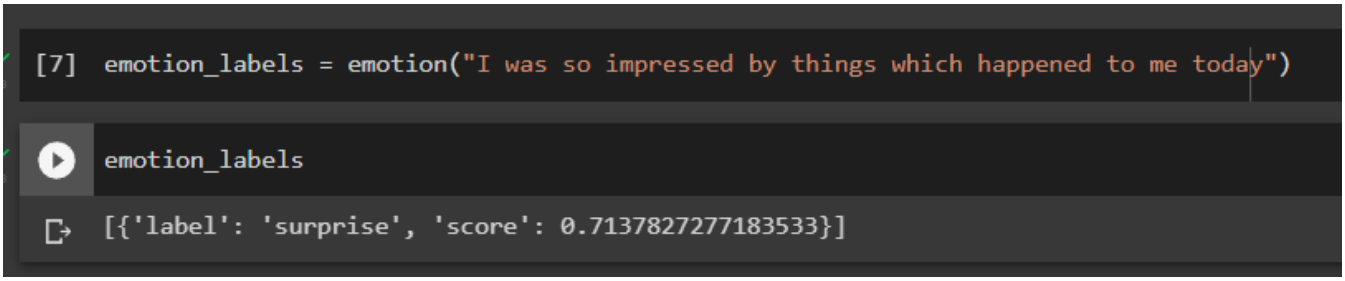
When you enter the username and hit “Request password reset link” button….

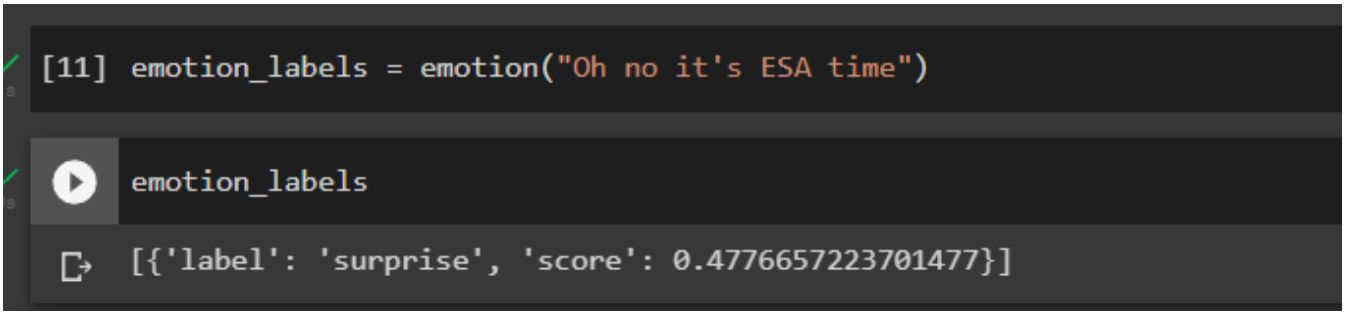


You get a message that a password token is sent to that particular mail id. You can reset your garble password through the token sent.



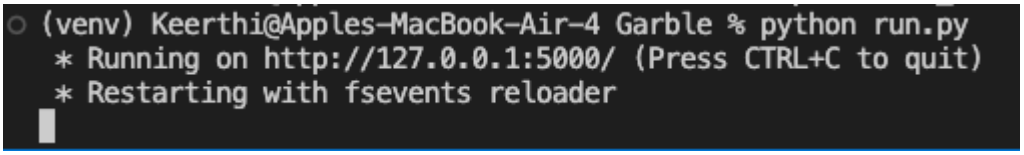
We used an online model to predict the mood of the user. Based on the text entered by the user, we can predict the mood.



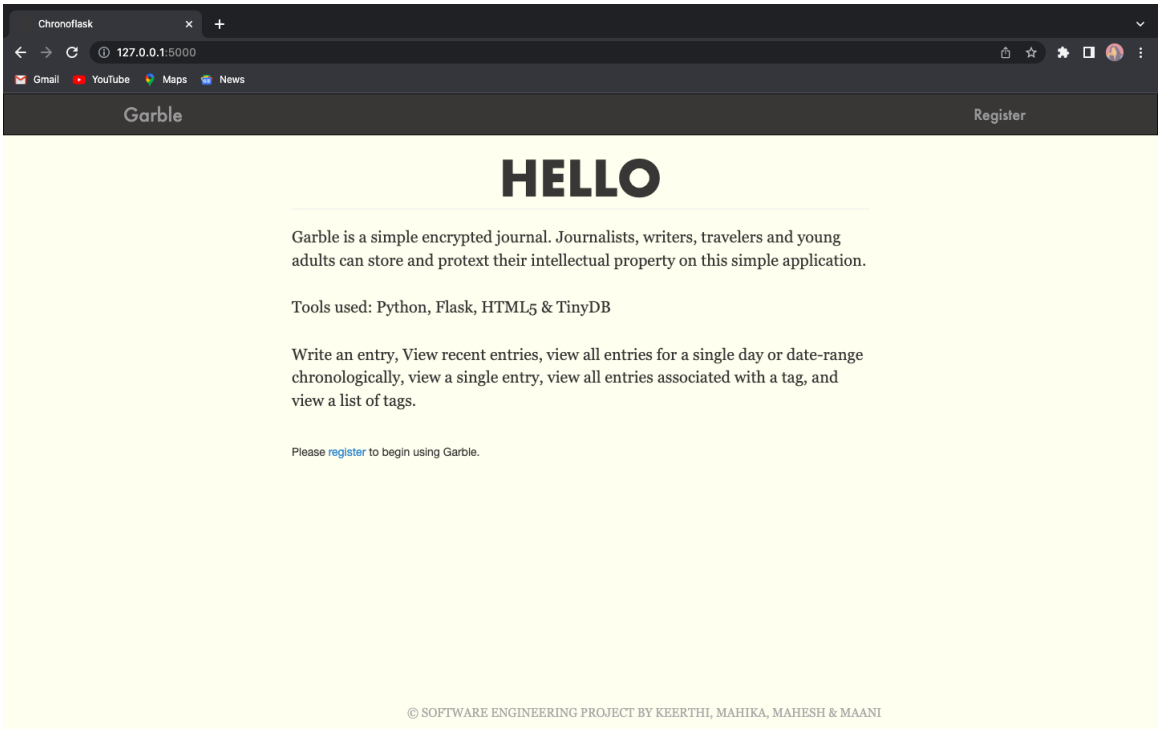


So based on text, the model predicts the mood of the author/user.

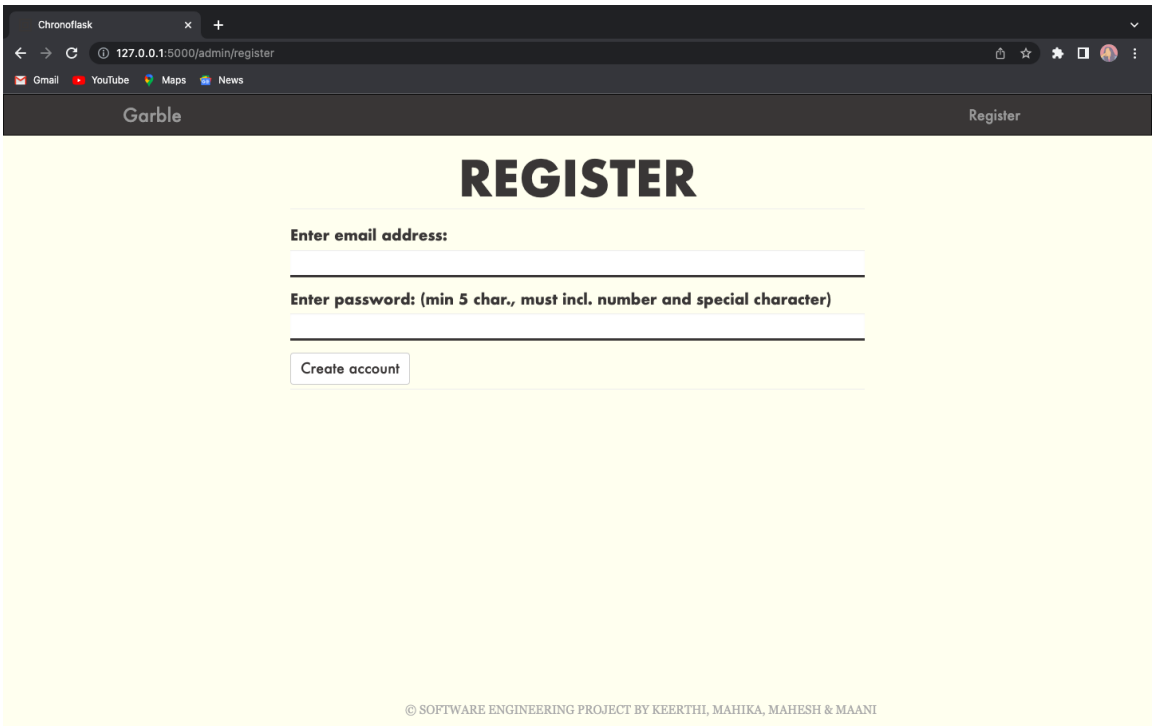
SCREENSHOTS OF OUTPUT



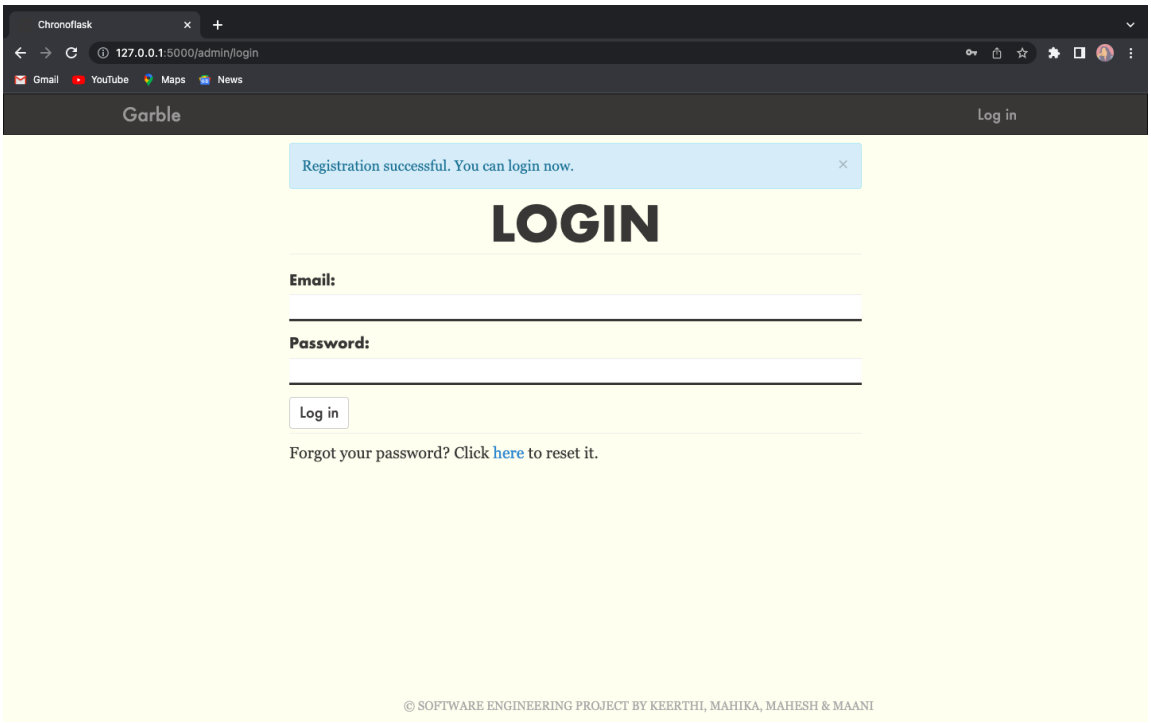
We run the python code on a virtual environment(venv)



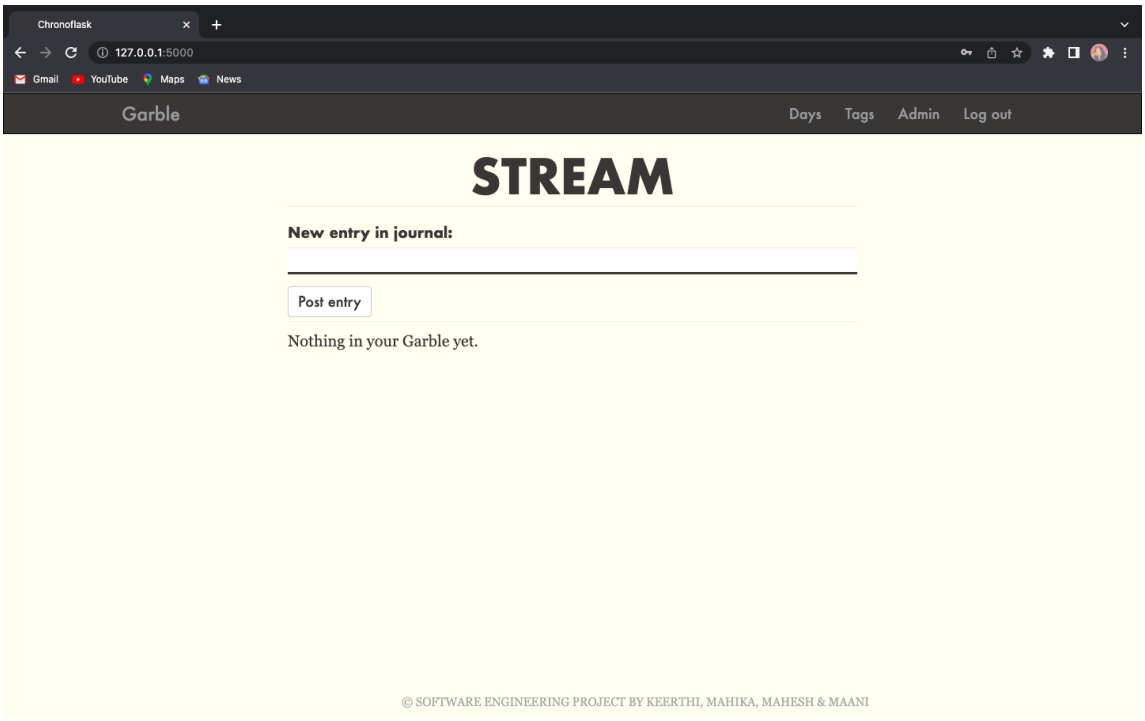
The landing page consists of a brief introduction to our project and the tools we have used. It also gives you a summary of the different functionality that is implemented in our project.



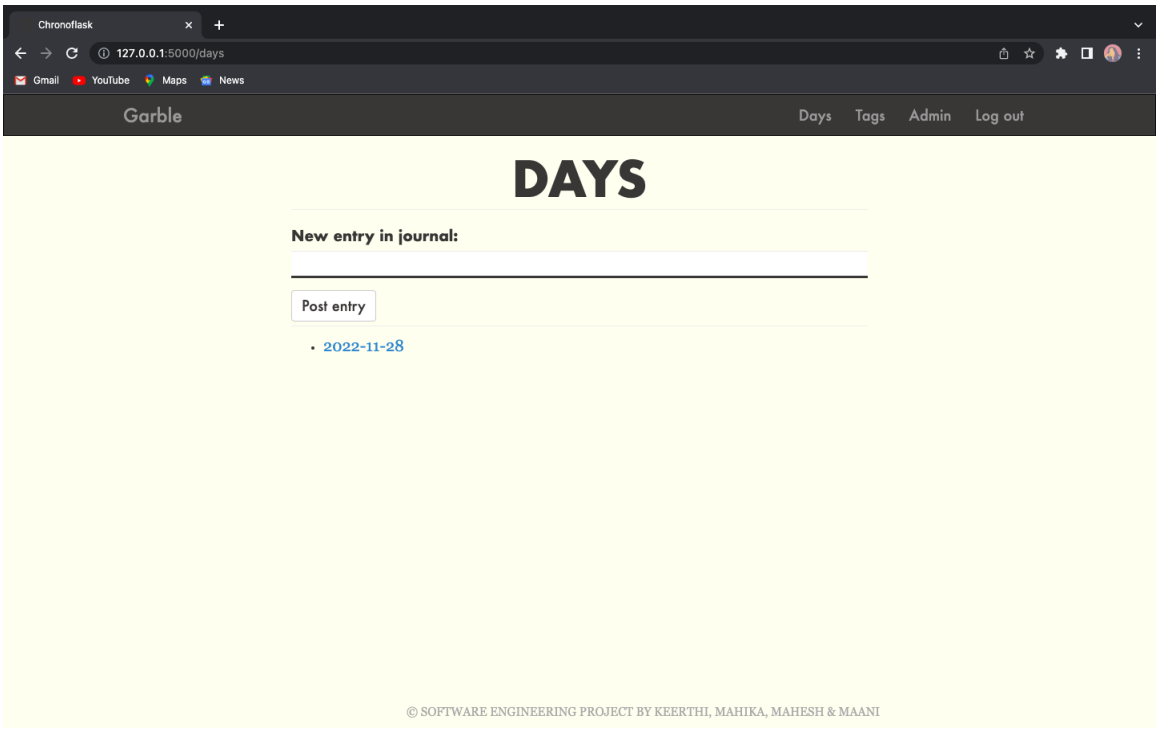
The first step in our project is to register a user.



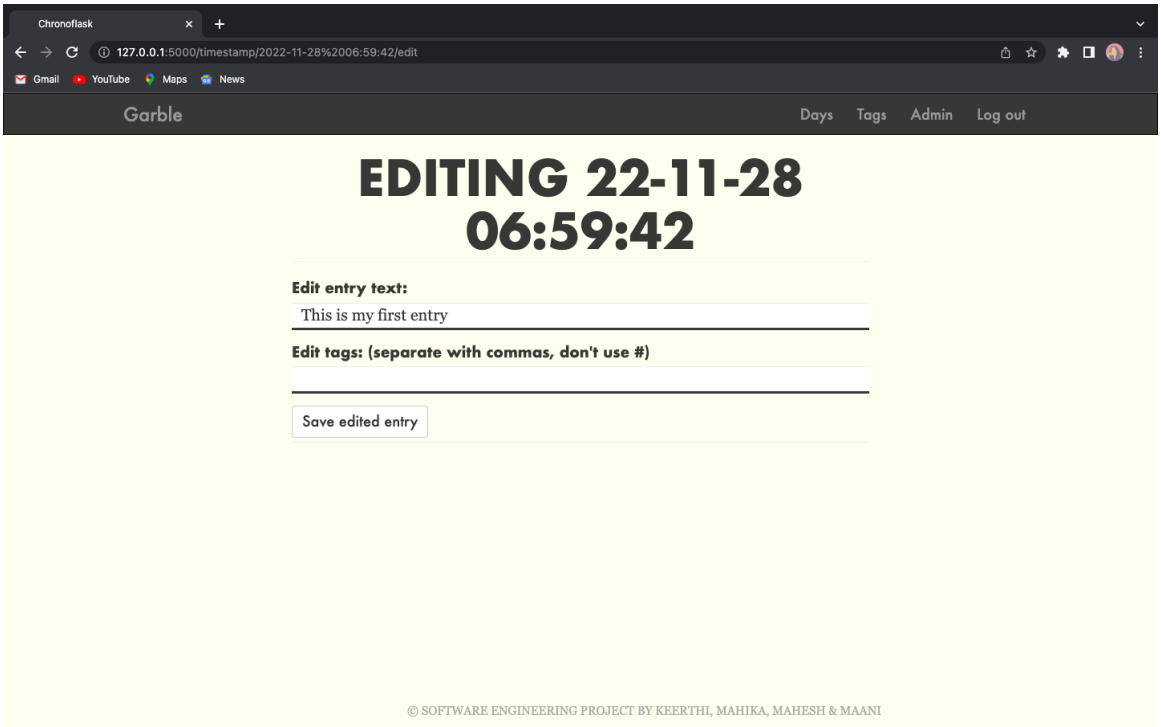
You get a flash message when all the details are entered correctly and the registration is successful.



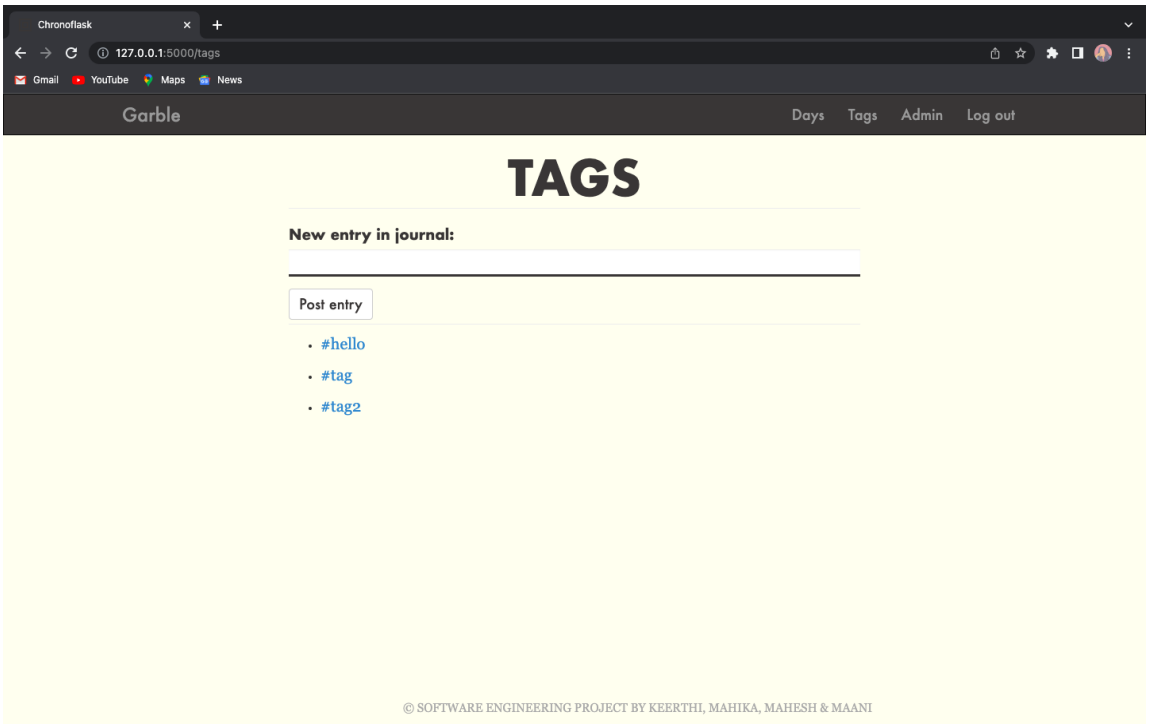
On successful login, you see the homepage which consists of the journal entry and the stream of journal entries.



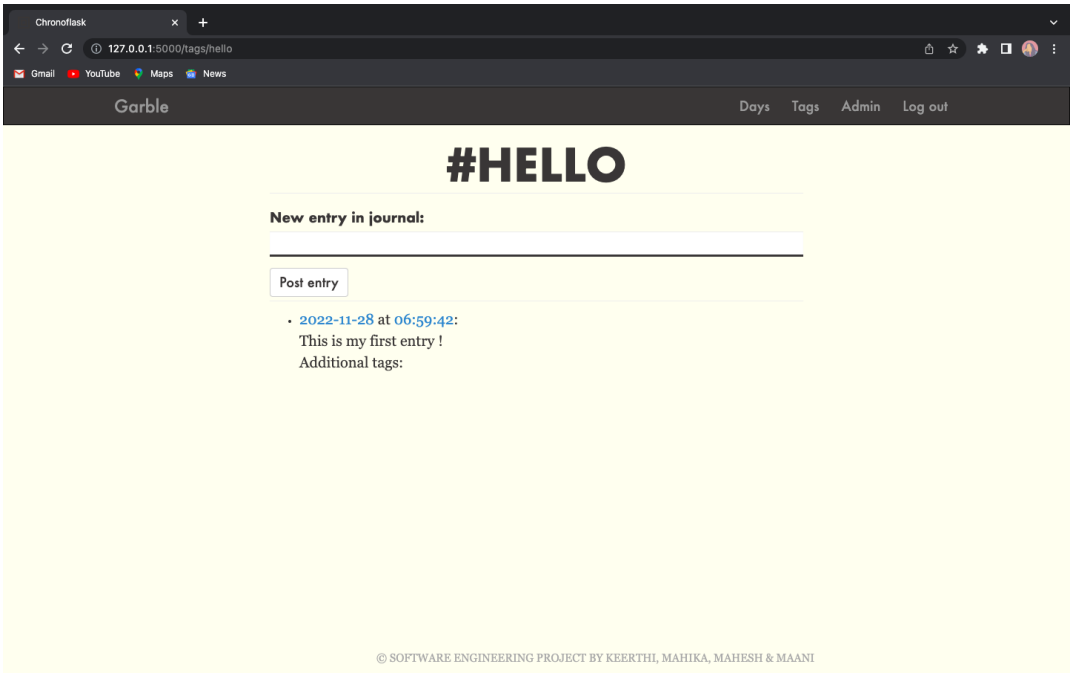
We now look at the "Days" implementation which will consist of a list of all entries made on a particular date.



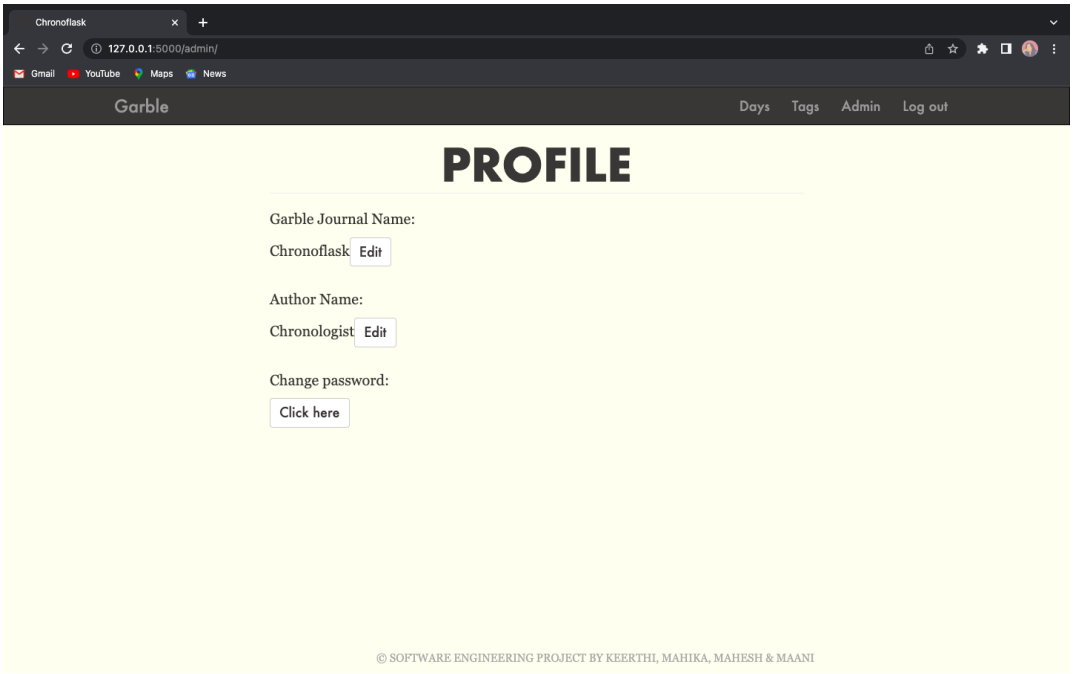
In the editing page we can either edit the text matter of the entry or the tags.



Now, when we look into the "Tags" page we can see all the tags used in all the entries of the journal.



We can go to the entry/entries that contain the tag in its entry.



In the Profile/Admin page we can change the name of the journal which will be reflected as the title of the site. We can also change the authors name of the journal to help personalize the feel of using this journal.

Encryption at backend:

```
(base) mahika@Mahikas-Air SE PROJ % python3 -u "/Users/mahika/SE PROJ/AES-Enc/aes.py"
Enter diary entry:Today was a good day. It was a sunday, and i was #happy
Enter key of 16 characters:mna123hg36/?@dg%
Encrypted text: b'\x95\x08\x05\xc7\xf8\x9cd\xbc\xc5\x0f\xa8\x886?\x980lq\r\x9a\x96ks\xb6\xb3\xc4.qd\xb0\xab\x9f~\x8f\x95\xb
5\x95*\xd5\xf4:\x9b\x1c\xeb\x12E\xb7\xd1\x02y\x86\xd4\xb3\xb7\x1e'
Decrypted text: Today was a good day. It was a sunday, and i was #happy
(base) mahika@Mahikas-Air SE PROJ %
```

Diary entry is stored in the database as encrypted text, and is displayed to the user in frontend after being decrypted.
AES python library is used, and the key is a 16-character string.

NLTK mood predicting module:

```
[7]  emotion_labels = emotion("I was so impressed by things which happened to me today")

     emotion_labels

     [{'label': 'surprise', 'score': 0.7137827277183533}]
```

```
[11] emotion_labels = emotion("Oh no it's ESA time")

     emotion_labels

     [{'label': 'surprise', 'score': 0.4776657223701477}]
```

Based on text, the model predicts the mood of the author/user.


PES
UNIVERSITY