Diary.com

Himashveta Kumar | Claire Brownell | Sierra Irving | Aayush Sasane

Chapter 1: Team Vision/Design

Introduction:

The goal of this project is to develop and deploy a cloud-based diary-writing web platform application. The app will allow users to write personal entries to themselves or to share their thoughts with the public via their profile. Entries can be locked with passwords or encryption so that no one but the user can access what they write. Users can also share access with other people via email for collaborative entries.

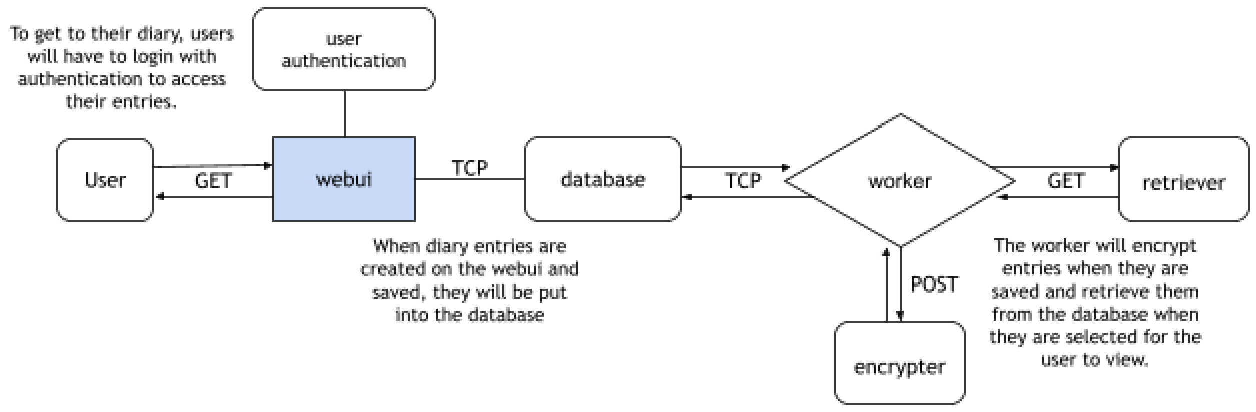
Features:

- **Personalized Profile UI:** Users can select a theme for a simple interface that allows them to rename/delete/share their entries in a Google style but with an emphasis on privacy for intimate diary entries.
- **Writing UI:** Opening entries transfer to a different editing UI for writing entries. Entries can be shared with other users via email so that they can also collaborate. Entries are not strictly limited to diary entries, as users can also use the interface to write poems, fiction, etc.
- **User-based Geo-location Sharing:** The website will allow users to use geolocation and profile matches to show other profiles that may have public entries of writing. In the future, users should be able to give comments on other users' public entries and there can be a liking system for popular entries in the local area.
- **Privacy:** The website emphasizes the importance of user privacy for intimate entries. Locked or private entries cannot be accessed by anyone except the user. Public entries can be seen by others and if any concerning content should arise, users may report the entry-writer for content violation.

Conclusion:

Diary.com (working title) isn't just another diary-writing platform; it's a sanctuary for personal expression and connection. Through our intuitive interface and emphasis on privacy, users can delve into their innermost thoughts without fear. It's more than just a place to jot down daily musings; it's a community where writers, poets, and storytellers can come together to share their creativity and experiences. With features like personalized profiles, collaborative entries, and geolocation sharing, Diary.com offers a unique blend of solitude and camaraderie. So whether you're penning a private reflection or sharing your work with the world, Diary.com is here to support and inspire you on your writing journey.

Design of the Infrastructure:


To get to their diary, users will have to login with authentication to access their entries.
user authentication
User
GET
webui
TCP
database
TCP
worker
GET
retriever
When diary entries are created on the webui and saved, they will be put into the database
POST
encrypter
The worker will encrypt entries when they are saved and retrieve them from the database when they are selected for the user to view.

Chapter 2: Team Design Implementation

Personalized Profile UI:

In our proposed software architecture, the frontend interface will utilize either HTML for static page structure, or JavaScript with frameworks such as React.js or Vue.js. These frameworks enable the design of responsive user interfaces that enhance user experience by making the diary application more interactive. For the server side, Node.js is chosen for backend development. This can help ensure efficient handling of simultaneous connections. For the database, either MongoDB or PostgreSQL will be used. MongoDB is useful for handling large volumes of unstructured data which will give high performance and easy scalability. PostgreSQL, is an efficient, open-source object-relational database system known for its reliability and performance when managing complex data types and relationships. To secure user data and ensure that only authorized users can access the application, Lightweight Directory Access Protocol (LDAP) is integrated for user authentication and authorization. This protocol allows the management of user credentials and permissions, providing a secure and efficient way to manage access to the application.

Writing UI:

For our development for writing the user interface, the application focuses on delivering an adequate user experience through different features, rich text editing, real time communication between the collaborators, and email notifications to inform users of any updates. Libraries such as Draft.js or Quill.js, which are well known for their comprehensive editing tools that allow users to format their text easily. This will make the content more engaging and visually appealing. To incorporate real-time collaboration, Socket.io is integrated. This enables instant communication between the collaborators. This ensures that all participants can see live updates, edits, and comments. This will enhance the collaborative experience by allowing team members to work together, regardless of their physical location. Our platform uses Nodemailer for email integration. This is a powerful module for Node.js applications that allows for sending email notifications and sharing entries directly from the platform. This feature will help streamline the process of distributing content as well as keeping collaborators informed about updates or changes.

Entry Storage:

To design the entry storage and geolocation attributes, the system will use either MongoDB or PostgreSQL as its database to provide strong and flexible storage for user entries along with their associated metadata. This approach allows for the optimization of data management based on the specific needs of the application. This approach allows the application to be functional but also provide users with a more personalized and context-aware experience.

Privacy:

Our project aims to develop and deploy a cloud-based application that prioritizes user privacy and security. Utilizing encryption technology, specifically Crypto.js, we will ensure that sensitive data, such as private entries, remains protected from unauthorized access. This encryption capability empowers users to lock their entries with passwords or encryption, granting exclusive access to their personal thoughts and reflections.

Moreover, our platform will feature a robust reporting system, backed by a database (MySQL), to uphold community standards and safety. Users can rest assured that their intimate entries are safeguarded, as locked or private entries cannot be accessed by anyone except the user. For those opting to share their thoughts publicly, a reporting mechanism will be in place to address any concerning content. This feature underscores our commitment to fostering a secure and respectful environment, where users can freely express themselves without compromising their privacy or encountering inappropriate content. Additionally, users will have the ability to share access with trusted individuals via email for collaborative entries, further enhancing the versatility and collaborative potential of our diary-writing platform.

Deployment:

Through this application, users will have the freedom to compose personal entries, intended solely for themselves, or opt to share their thoughts with the public via their profile. Emphasizing the users privacy and security.

Furthermore, to enhance collaboration and shared experiences, users can extend access to trusted individuals through email invitations, enabling collaborative entries. Leveraging technology, our deployment strategy incorporates Docker for containerization, streamlining the deployment process and promoting consistency across different environments. Additionally, Kubernetes will serve as the backbone for container orchestration, enabling efficient management and scalability of our application's infrastructure.

By integrating these software solutions into our project, we aim to deliver a reliable diary platform that empowers users to document their thoughts securely and engage with others in a collaborative and meaningful manner.

Chapter 3: Intermediate Milestones

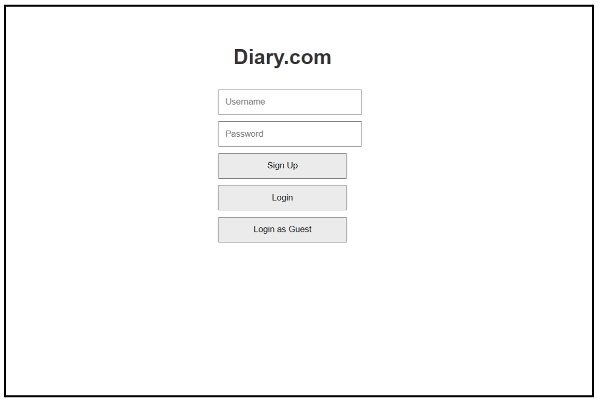
Current Progress on Diary.com:

At this point, our group's main focus is on creating the front-end aspect of the website to ensure that we have something to demonstrate as a viable user experience when people use our product to write personal entries on their accounts. A majority of development has been spent on the WebUI aspect and the development of databases to store and retrieve document entries written by users. Due to the inexperience of the group in such technologies, a lot of time has been invested in experimenting with different frameworks to determine what web framework, front-end and back-end technologies, and database frameworks would work best for our project.

WebUI and Front-End Development

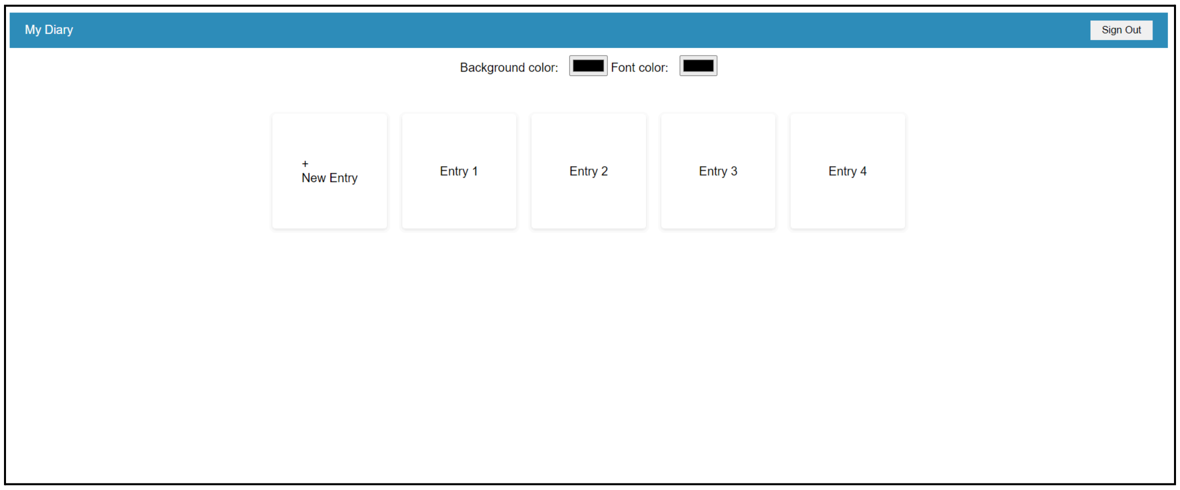
The WebUI allows the user to view and interact with our website. The index.html is the login page which will prompt the user for their credentials; if they are not a member there is an option to sign up. Additionally, if the user does not wish to create an account, there is a sign in as a guest option which will allow them to use our website, but it will not save their entries. This can act as a free trial to test it out before signing up. The diary.html contains html script as well as JavaScript functions which will make the page interactive. This includes the save entry, sign out, and new entry buttons.

Index.html page:
This will prompt the user to login with their credentials, sign up, or continue as a guest.



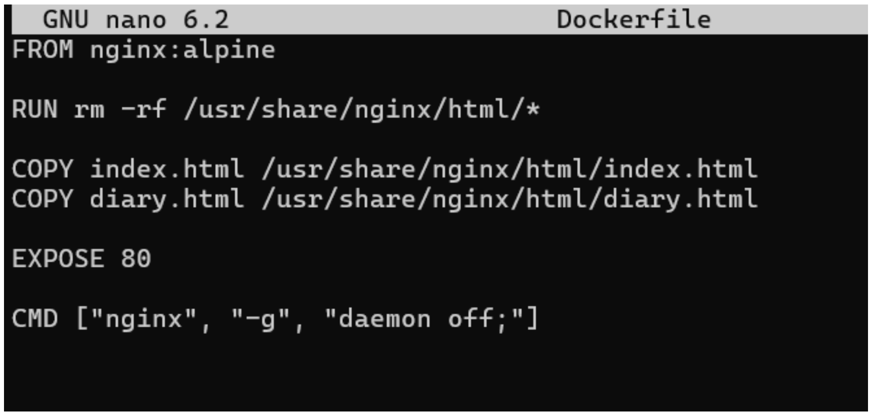
Diary.html page:

From the diary page, the user has the option to create their new entries or view previous ones. The new entry button will take them to a text editor which will allow them to write whatever they would like. Once saved, the entry will be shown on the page which will let the user revisit and edit. To make the website more customizable, there are two drop down menus that allow the user to change the font and background colors to their liking. There is a sign out button in the top right corner which will take you back to the login page once you log out.



Dockerfile for WebUI deployment:

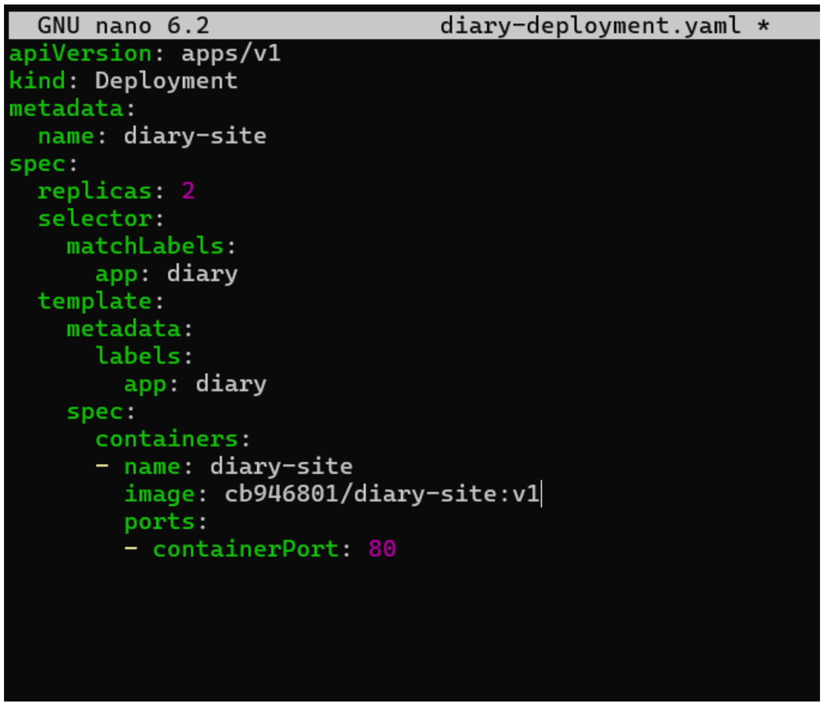
The Dockerfile uses nginx:alpine as the base image. It then clears out any pre-existing html files in the default nginx server directory to make room to copy our index.html and diary.html files. The default port is set to 80 so it can accept any requests.



```
  GNU nano 6.2                         Dockerfile
FROM nginx:alpine

RUN rm -rf /usr/share/nginx/html/*

COPY index.html /usr/share/nginx/html/index.html
COPY diary.html /usr/share/nginx/html/diary.html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Diary-deployment.yaml:

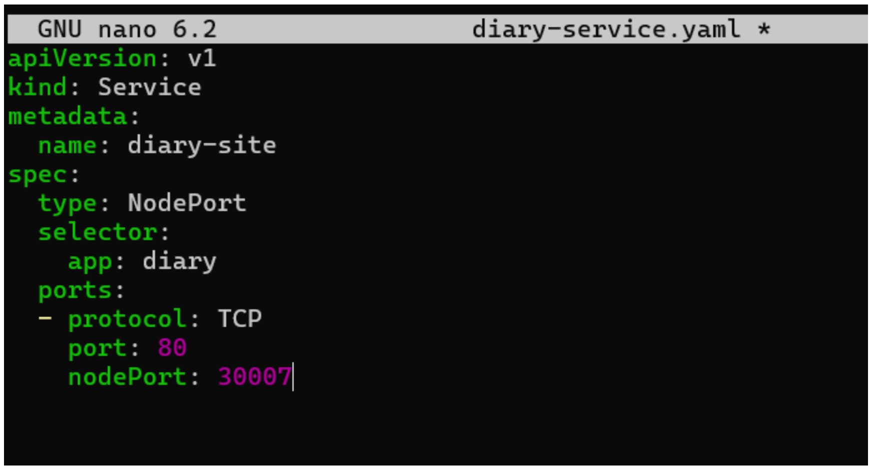
Within the diary-deployment.yaml, it is configured accordingly for Kubernetes to deploy the application across the cluster. It will tell Kubernetes to create pods based on the image that was built from the dockerfile above. If any instances were to fail or be deleted, it will automatically replace them. The deployment uses labels to manage which pods belong to it, facilitating updates and rollbacks.



```
  GNU nano 6.2                    diary-deployment.yaml *
apiVersion: apps/v1
kind: Deployment
metadata:
  name: diary-site
spec:
  replicas: 2
  selector:
    matchLabels:
      app: diary
  template:
    metadata:
      labels:
        app: diary
    spec:
      containers:
      - name: diary-site
        image: cb946801/diary-site:v1
        ports:
        - containerPort: 80
```

Diary-service.yaml:

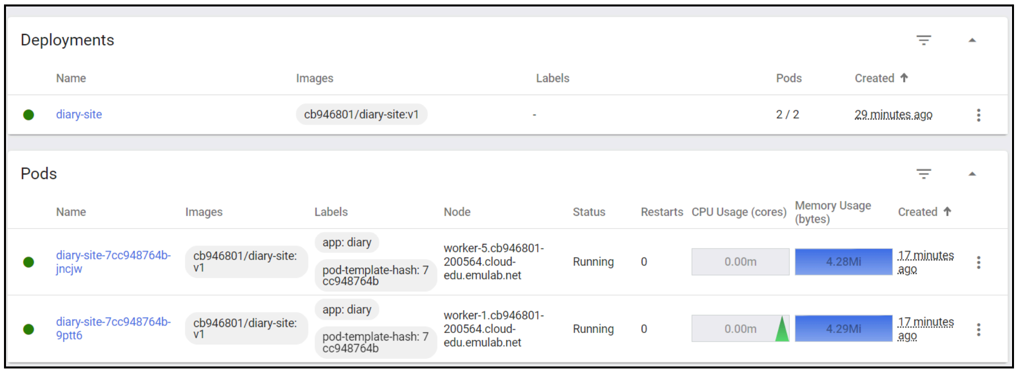
The diary-service.yaml is configured to Kuberntes. It serves as an IP address to remain constant as pods are created or deleted to maintain traffic to our applications pods. Nodeport was chosen as the type of service to make it accessible on a specific port of every node within the cluster.



```
  GNU nano 6.2                    diary-service.yaml *
apiVersion: v1
kind: Service
metadata:
  name: diary-site
spec:
  type: NodePort
  selector:
    app: diary
  ports:
  - protocol: TCP
    port: 80
    nodePort: 30007
```

Successful deployment:

The deployment, pods, and replicas were successfully created from the Kubernetes dashboard.



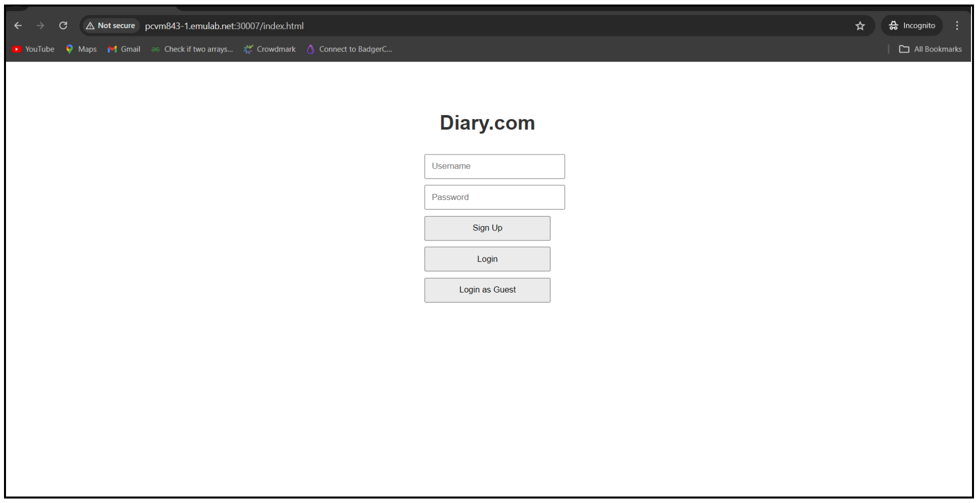
Deployments

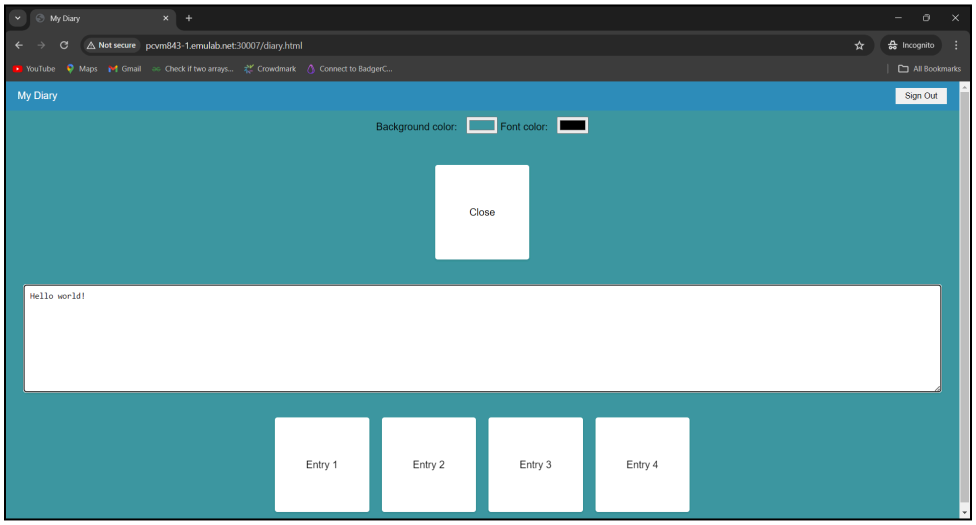
	Name	Images	Labels	Pods	Created ↑
●	diary-site	cb946801/diary-site:v1	-	2 / 2	29 minutes ago

Pods

	Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
●	diary-site-7cc948764b-jncjw	cb946801/diary-site:v1	app: diary pod-template-hash: 7cc948764b	worker-5.cb946801-200564.cloud-edu.emulab.net	Running	0	0.00m	4.28Mi	17 minutes ago
●	diary-site-7cc948764b-9ptt6	cb946801/diary-site:v1	app: diary pod-template-hash: 7cc948764b	worker-1.cb946801-200564.cloud-edu.emulab.net	Running	0	0.00m	4.29Mi	17 minutes ago

The website is accessible via cloudlab experiment link:
http://pcvm843-1.emulab.net:30007/index.html





Next Steps:

- **Node.js:** This could help us create a web server to serve the html files to the browser.
  - Allow us to use dynamic data, not just static (HTML).
  - This will help us build the back-end part of our project to eventually establish communication between our PostgreSQL database.
- **Bootstrap:** Create a structured and well designed website to make it more visually pleasing for the users.
  - Add more interactive buttons to the website such as share, make public or private, and create a working save button for users to retrieve old entries.
  - Create another page for users to sign up and save their credentials to use our website.

Issues:

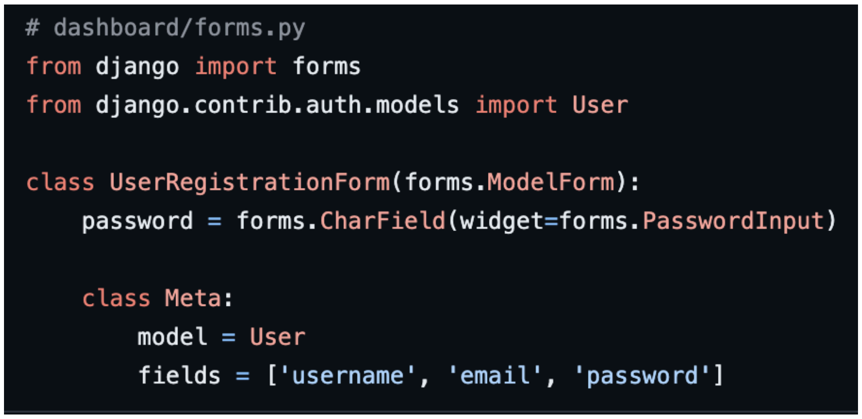
- Since the back-end of the project is still being worked on, some buttons do not have functionality.
  - Example: the save button does not work yet, the entries cannot be saved.

User Authentication

In our project, we initially planned to implement user authentication using the LDAP framework due to our prior experience with LDAP directories from previous projects. However, after careful consideration, we decided to switch to Django for user authentication, primarily to seamlessly connect with the WebUI and our PostgreSQL server.

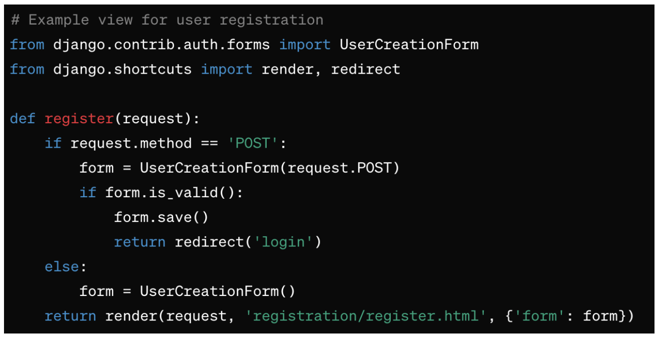
Decision to Use Django: We made the decision to utilize Django for user authentication as it provides a robust authentication system out of the box and integrates well with PostgreSQL. LDAP would still work but Django seemed like a better selection with the WebUI choices.

Setting Up Django Project: We began by setting up our Django project, including creating the necessary files and directories using the django-admin command and defined models in Django to represent user information and document entries, ensuring a clear structure for data storage in our PostgreSQL database.



```python
# dashboard/forms.py
from django import forms
from django.contrib.auth.models import User

class UserRegistrationForm(forms.ModelForm):
    password = forms.CharField(widget=forms.PasswordInput)

    class Meta:
        model = User
        fields = ['username', 'email', 'password']
```

We began working on functionality to associate user documents with their respective accounts, ensuring seamless integration with PostgreSQL.We also began working on functionality to associate user documents with their respective accounts, ensuring seamless integration with PostgreSQL.



```python
# Example view for user registration
from django.contrib.auth.forms import UserCreationForm
from django.shortcuts import render, redirect

def register(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('login')
    else:
        form = UserCreationForm()
    return render(request, 'registration/register.html', {'form': form})
```

Challenges and Issues:

- **Configuration Errors:** We faced some configuration errors while setting up PostgreSQL with Django, particularly with database connection settings.
    - Through troubleshooting and consulting Django documentation, we will be able to resolve these issues.
- **Complete User Authentication:** We plan to finish implementing all necessary views and templates for user authentication, ensuring a smooth user experience.
- **Integrate with WebUI:** We will integrate the Django authentication system with our WebUI, allowing users to log in and access their documents seamlessly.
- **Document Integration with PostgreSQL:** We will continue working on integrating user documents with PostgreSQL, ensuring proper data storage and retrieval.

Security in front-end and database connection

Even after logging in we want to ensure a secure environment for our users to write their entries in, so privacy is of the utmost importance. Users will be able to decide between encrypting their entries and password protecting them to give them the peace of mind that their private entries are safe and remain for their eyes only.

Encryption Options

For users who decided to encrypt Crypto.js and Forge will be used for an AES (Advanced Encryption Standard) symmetric key that will be utilized to encrypt the entries that are typed. Users who choose to encrypt their entries will be given a token to decrypt their entries for an added level of privacy.

Password Protection with Django

Users who decide to password protect their entries will be given the option to choose their password when they create the entry. Each entry will be able to have a different password chosen by the user as long as it is eight characters long including at least one number and one special character. We have not implemented that but for now we have an entry form on Django for password protecting the contents of an entry as below:

```
class EntryForm(forms.Form):
    content = forms.CharField(widget=forms.Textarea)
    password = forms.CharField(widget=forms.PasswordInput)
```

Secure Connection with the Database

When any entries are saved and moved to the database, the connection will be secure by establishing channels using TLS. Again, we want our users to be able to have confidence in our ability to keep their data safe, so we are focusing a lot of attention creating a safe and secure database for these entries. This has been started with a django script using Fernet encryption to encrypt data before storing them in the database.

```
from cryptography.fernet import Fernet

key = Fernet.generate_key()
cipher_suite = Fernet(key)
encrypted_token = cipher_suite.encrypt(b"my_secret_token")
```

Next Steps:

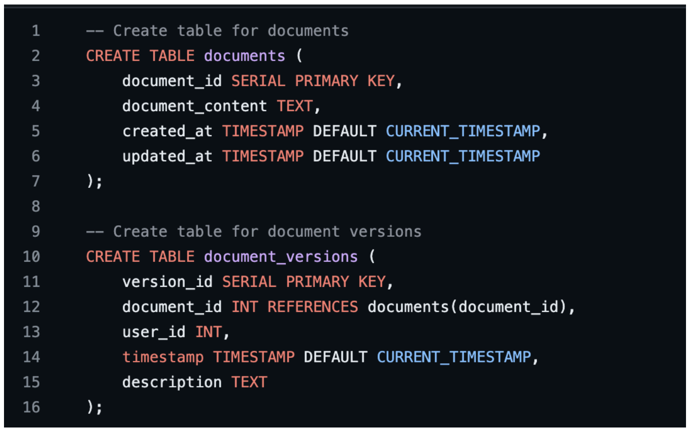
As of now, we're still in the preliminary stages of this process, focusing on laying the groundwork for implementing advanced security features in our Django application. Our immediate next steps involve conducting thorough research and planning to finalize the architecture and design for integrating password protection, token-based decryption, and database encryption.

PostgreSQL Database Development

The database development needs to safely store user data and facilitate efficient management of document entries and their versions. We have initiated the setup of a PostgreSQL database within a Docker container orchestrated by Kubernetes. The focus moving forward is on establishing robust authentication and authorization mechanisms to ensure data security. Additionally, we are integrating the database with a Flask backend worker to facilitate seamless connectivity with the frontend WebUI, particularly the user dashboard and document management

Database Initial Setup:

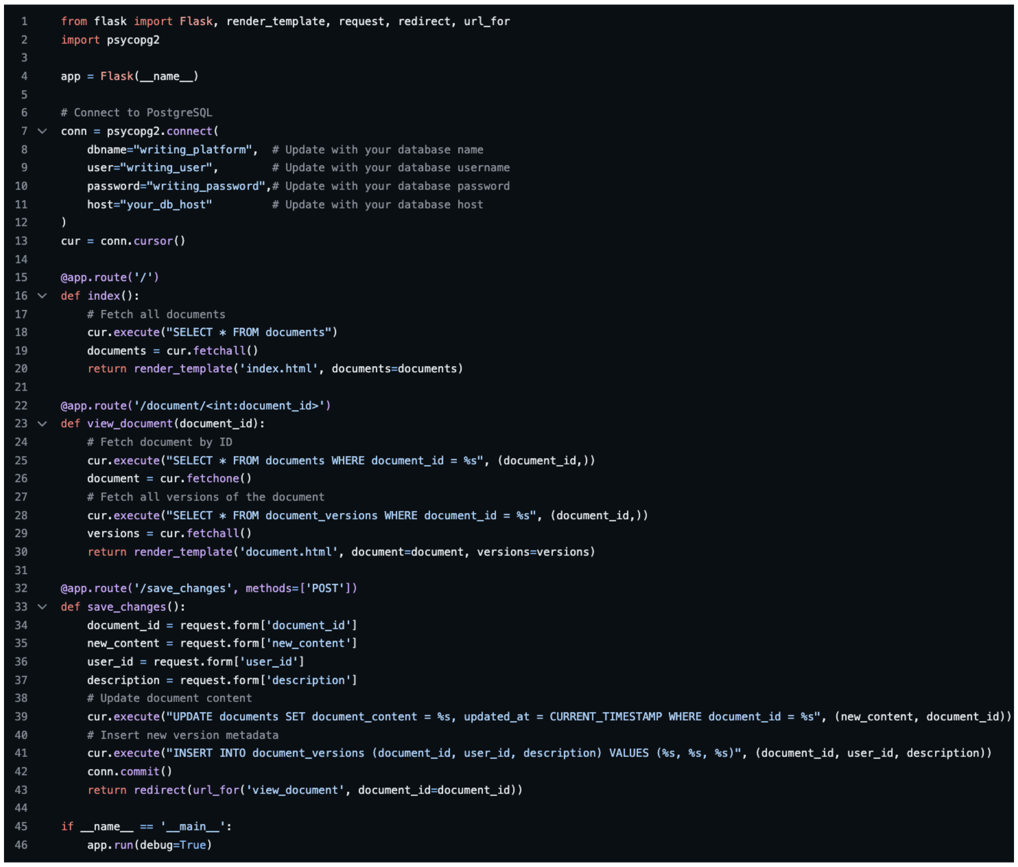
In the initial stages, we have defined the database schema to accommodate user entries and version history. The table setup stores metadata, versions, and other information related to documents. Below is the init.sql file with the table setup within the Dockerfile of the PostgreSQL image:



```sql
-- Create table for documents
CREATE TABLE documents (
    document_id SERIAL PRIMARY KEY,
    document_content TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create table for document versions
CREATE TABLE document_versions (
    version_id SERIAL PRIMARY KEY,
    document_id INT REFERENCES documents(document_id),
    user_id INT,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    description TEXT
);
```

Flask Backend Integration:

To connect between the frontend and the database, we have written a Flask backend worker. This worker serves as the intermediary layer responsible for handling requests from the user dashboard and managing interactions with the PostgreSQL database. Currently, the integration is in progress, and we aim to refine it further to ensure smooth communication between the frontend and backend components. The file is shown below:



```python
from flask import Flask, render_template, request, redirect, url_for
import psycopg2

app = Flask(__name__)

# Connect to PostgreSQL
conn = psycopg2.connect(
    dbname="writing_platform",  # Update with your database name
    user="writing_user",        # Update with your database username
    password="writing_password",# Update with your database password
    host="your_db_host"         # Update with your database host
)
cur = conn.cursor()

@app.route('/')
def index():
    # Fetch all documents
    cur.execute("SELECT * FROM documents")
    documents = cur.fetchall()
    return render_template('index.html', documents=documents)

@app.route('/document/<int:document_id>')
def view_document(document_id):
    # Fetch document by ID
    cur.execute("SELECT * FROM documents WHERE document_id = %s", (document_id,))
    document = cur.fetchone()
    # Fetch all versions of the document
    cur.execute("SELECT * FROM document_versions WHERE document_id = %s", (document_id,))
    versions = cur.fetchall()
    return render_template('document.html', document=document, versions=versions)

@app.route('/save_changes', methods=['POST'])
def save_changes():
    document_id = request.form['document_id']
    new_content = request.form['new_content']
    user_id = request.form['user_id']
    description = request.form['description']
    # Update document content
    cur.execute("UPDATE documents SET document_content = %s, updated_at = CURRENT_TIMESTAMP WHERE document_id = %s", (new_content, document_id))
    # Insert new version metadata
    cur.execute("INSERT INTO document_versions (document_id, user_id, description) VALUES (%s, %s, %s)", (document_id, user_id, description))
    conn.commit()
    return redirect(url_for('view_document', document_id=document_id))

if __name__ == '__main__':
    app.run(debug=True)
```

Kubernetes Deployment and PostgreSQL Server Setup:

For our project's backend infrastructure, we orchestrated the deployment of a PostgreSQL server within a Kubernetes cluster. Using YAML files, we defined both the deployment and service configurations to ensure seamless instantiation and accessibility of the database. The deployment YAML file specified the container image, resource requirements, environment variables, and volume mounts necessary for PostgreSQL operation. Concurrently, the service YAML file facilitated internal exposure of the PostgreSQL server within the cluster, defining ports and selectors for traffic routing. As a result, the PostgreSQL server was successfully instantiated within the Kubernetes environment, equipped with the necessary tables as defined in the init.sql file, laying the foundation for secure and efficient data storage.

Next Steps:

- **User Authentication:** One of the top priorities is to strengthen user authentication mechanisms within the Flask backend between the WebUI and the PostgreSQL server.
    - This includes implementing secure login and session management functionalities to authenticate and authorize users effectively.

- **Enhanced Functionality:** We will continue to expand the capabilities of the PostgreSQL server to serve as a comprehensive backend solution for the website.
    - This involves developing functionalities to assist users in managing and tracking their document entries efficiently.

Backend Workers for Database Communication/Connectivity

In our project, establishing proper backend workers for database communication and connectivity is vital to ensure a correct connection between the frontend WebUI and the PostgreSQL database. We have initiated the development of Flask backend workers to serve as the intermediate layer responsible for handling requests from the user dashboard and managing interactions with the PostgreSQL database.

Current Progress:

At present, the integration of Flask backend workers with the PostgreSQL database is underway. We are focusing on developing endpoints to handle user authentication, document retrieval, creation, modification, and deletion. Additionally, we are implementing robust error handling and security measures to safeguard sensitive user data during transit and storage.

Next Steps:

Moving forward, our priorities include:

- **Authentication Enhancement:** Strengthening authentication mechanisms within the Flask backend to ensure secure user login and session management.
- **Error Handling and Validation:** Implementing comprehensive error handling and input validation to prevent data inconsistencies and enhance system reliability.
- **Optimization:** Optimizing backend operations for improved performance and scalability, ensuring smooth handling of concurrent user requests.

By prioritizing these next steps, we aim to establish a robust backend infrastructure that effectively supports the functionality and security requirements of Diary.com.

Kubernetes Deployment

So far, the WebUI and the PostgreSQL server have been deployed with yaml files but it is not a fully connected system as of this moment. The main focus in terms of deployment moving forward will be to organize the pods with applicable nodes. Communication between deployments is a primary objective to be completed soon before the next milestone of the project.

Issues and Points of Improvement

In our project, we encountered several issues and identified points of improvement, with a key focus on enhancing security and optimizing database management. We realized that rather than prioritizing bells and whistles, it was essential to concentrate on implementing robust security measures.

Encryption

- One significant aspect was paying meticulous attention to our database setup and management, particularly regarding encryption.
- We identified an opportunity to improve our encryption strategy by encrypting data at the time of input, rather than retroactively.

Token System

- Additionally, we considered implementing a token-based system to provide users with secure access to decrypt their data, thereby enhancing overall data security.
- Moving forward, we aim to prioritize security measures and refine our database management practices to ensure the utmost protection of user data.