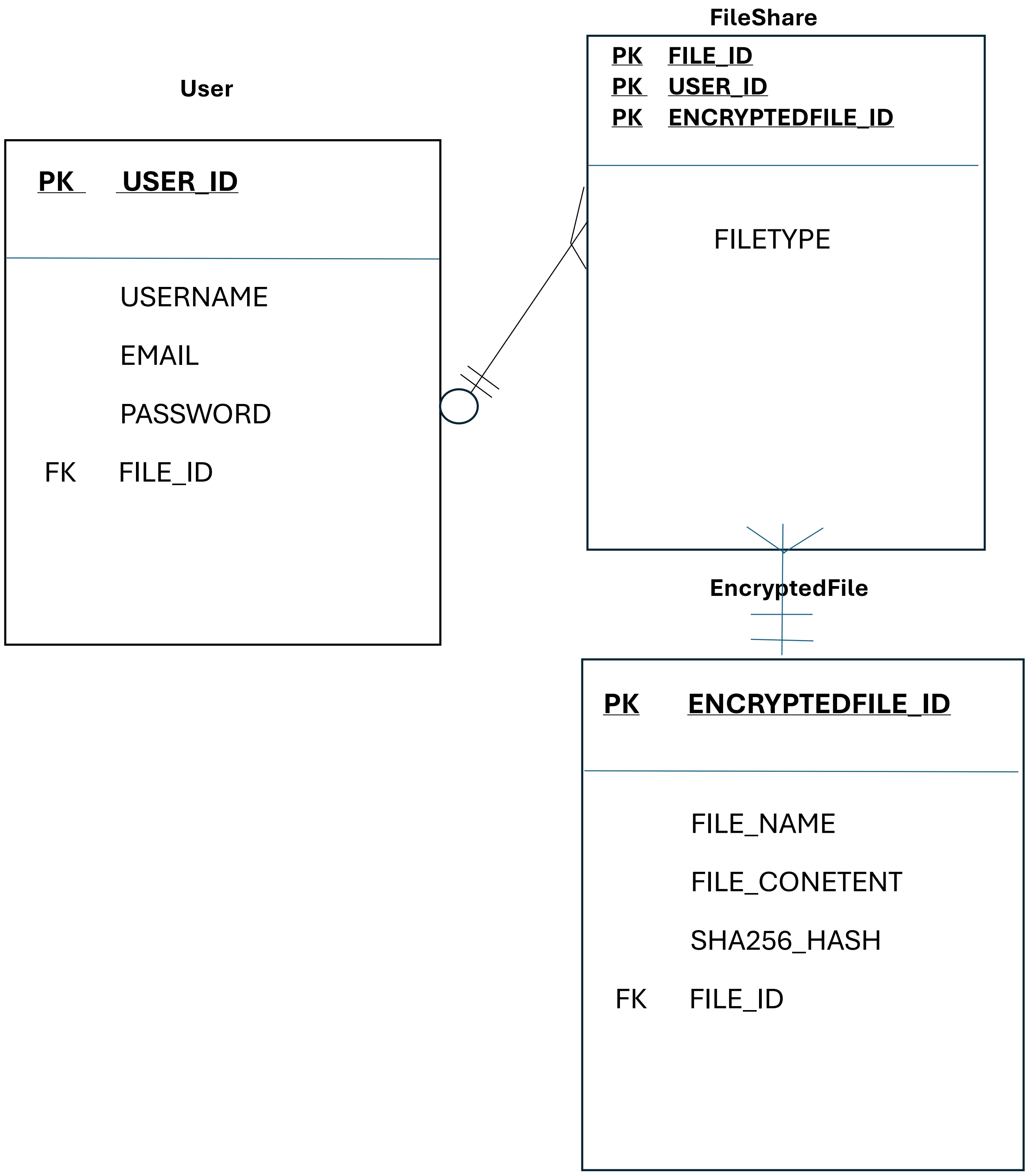
Report

Assumptions:

1. File Types and Size: File types will be standard (pdf,txt,jpg,jpeg,mp4,mp3,docx), and no specific size limitations are mentioned.
2. Encryption: Symmetric encryption will be used (same key for encryption and decryption), with the encryption key securely stored as an environment variable.
3. User Permissions: Users can access only their own files or those shared with them, with no additional roles or privileges beyond this.
4. File Sharing: Files can be shared with other users, and file-sharing functionality will be seamless.
5. Data Integrity: The sha256_hash field will be used for file integrity verification, ensuring that encrypted files haven't been altered.
6. Authentication: User registration and authentication will be handled via Django's built-in system.
7. File Storage: Encrypted files will be stored securely in the database, and file content will be encrypted before storage.

We’ve also created the **Entity-Relationship Diagram (ERD**) and file encryption algorithm to map out the database structure and encryption/decryption flow.



Relationships and Cardinalities

1. **User → EncryptedFile (Owner)**:
    - **Relationship Strength**: Strong
    - **Cardinality**: 1
        - One User owns many EncryptedFile instances.
2. **EncryptedFile ↔ User (Shared)**:
    - **Relationship Strength**: Weak (via FileShare bridge table)
    - **Cardinality**: M
        - One EncryptedFile can be shared with multiple User instances.
        - One User can receive/shared multiple EncryptedFile instances.
3. **FileShare (Bridge)**:
    - Links the EncryptedFile

Algorithm

1. User Registration

Algorithm: Register a new user

Accept a username, email, and password from the user.

->Validate the inputs:

- Ensure the username contains only alphanumeric characters and underscores.
- Check that the email is properly formatted.
- Verify the password meets security requirements (length, numbers, uppercase).
- Check if the username or email already exists in the database.

->If all validations pass:

Save the user information in the database.

Encrypt the password using Django's built-in password hashing.

Provide a success message or redirect the user.

If validations fail, display an appropriate error message.

2. User Login

Algorithm: Authenticate a user

Accept a username and password from the user & check the database for a matching username.

->If the username exists:

- Verify that the provided password matches the hashed password stored in the database.

->If the credentials are correct:

- Log the user in and redirect to their dashboard or home page.
- If the credentials are incorrect, show an error message.

3. File Upload

Algorithm: Upload and encrypt a file

Accept a file and metadata from the user.

- Check if the user is logged in and authorized to upload files.
- Encrypt the file using AES encryption:
- Generate a symmetric encryption key.
- Encrypt the file content using the key.
- Save the encrypted file to disk.
- Store the file metadata (including the encrypted key) in the database.
- Provide a confirmation message that the file was uploaded successfully.

4. File Download

Algorithm: Decrypt and download a file

Accept a file download request from a user.

- Check if the user is authorized to access the file:
- Verify if they own the file or have been granted access by the owner.

- Retrieve the encrypted file and its encryption key from the database.
- Decrypt the file using the stored key.
- Serve the decrypted file for download.
- If the user is unauthorized, deny access with an appropriate message.

5. File Sharing

Algorithm: Share a file with another user

Accept a request to share a file:

- File identifier.
- The username of the recipient.
- Check if the requesting user owns the file.
- Validate the recipient's username in the database.
- Create a sharing record in the database:
- Associate the recipient's user ID with the file.
- Notify the recipient (optional).
- Confirm the file-sharing action with the owner.

6. File Integrity Check

Algorithm: Verify file integrity

->During file upload:

- Compute the SHA-256 hash of the original file content.
- Store the hash in the database.
- During file download:
- Compute the SHA-256 hash of the decrypted file content.
- Compare the computed hash with the stored hash.
- If the hashes match:

->Proceed with the download.

- If the hashes don't match:
- Reject the download and notify the user about potential tampering.

7. Unauthorized Access Handling

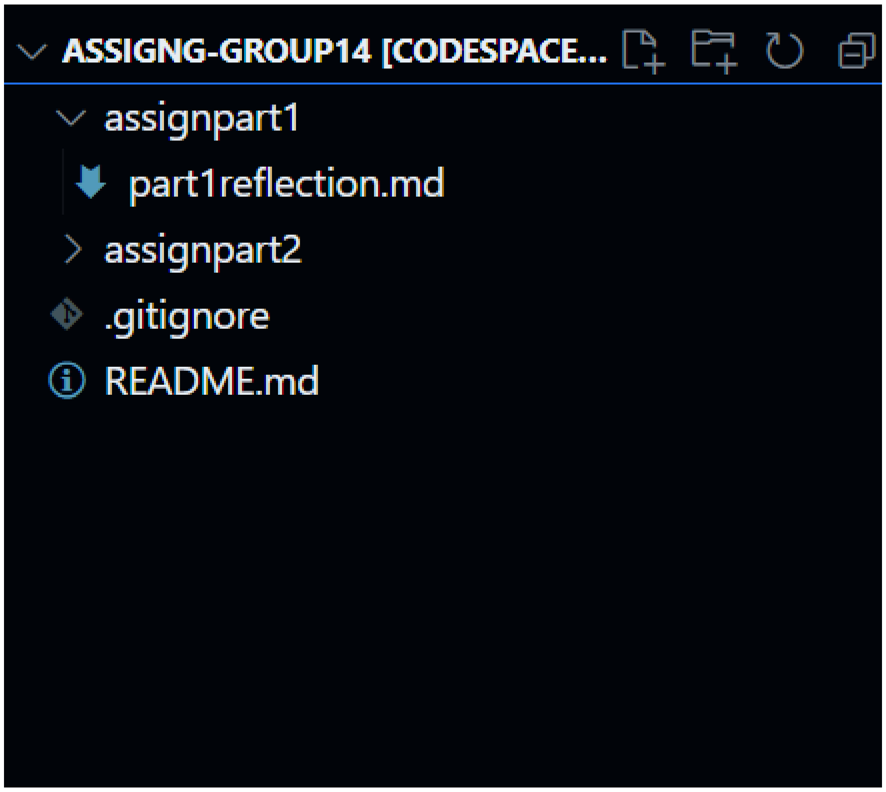
Algorithm: Handle unauthorized access

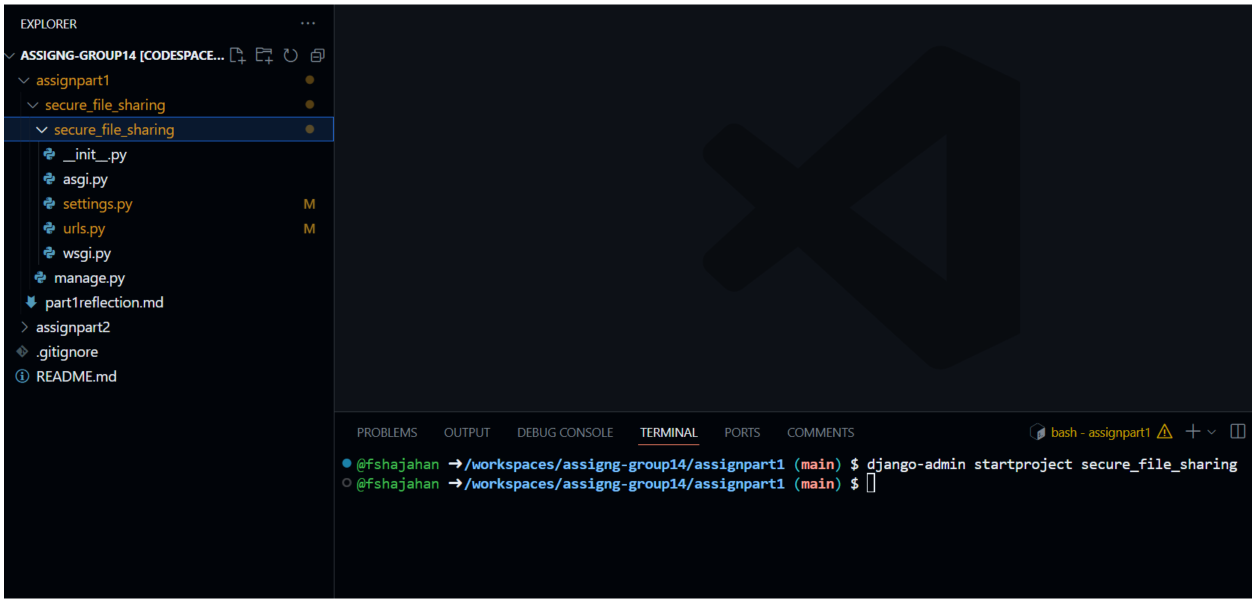
- For any file-related request:
- Check if the user is logged in.
- Check if the user is authorized to access the requested file.
- If unauthorized:
- Log the access attempt.
- Deny access with an error message or redirect to a login page.

With this assumptions, and designwe started our project

1.Setting up the project

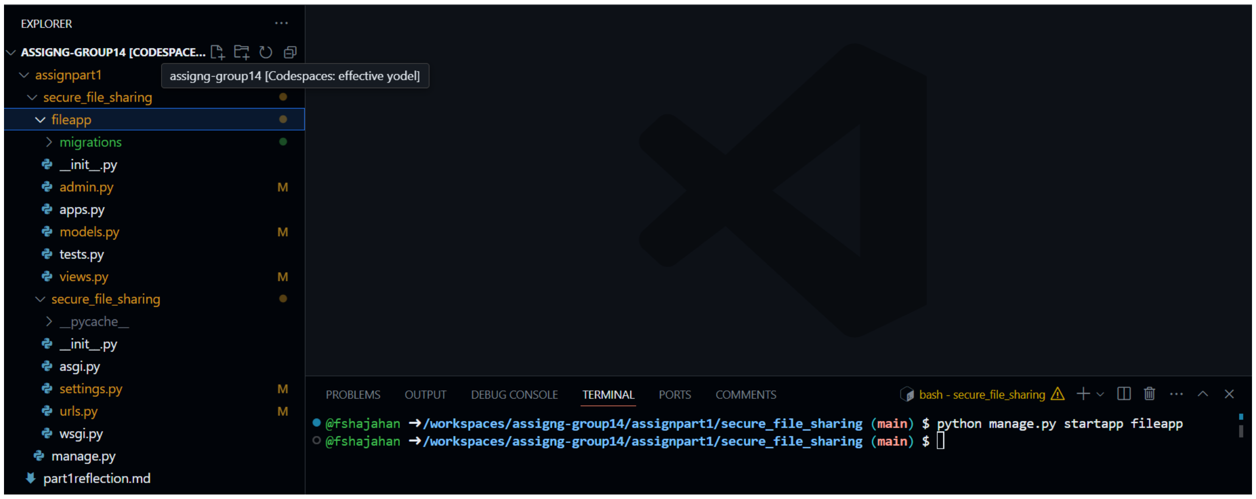
- First, we created a new Django project using the command django-admin startproject secure_file_sharing. This initialized the project structure, including essential files like settings.py, urls.py.





```
@fshajahan →/workspaces/assigng-group14/assignpart1 (main) $ django-admin startproject secure_file_sharing
@fshajahan →/workspaces/assigng-group14/assignpart1 (main) $ 
```

- We then created a new app within the project using python manage.py startapp fileapp. This helped in organizing the file management functionality into its own module.

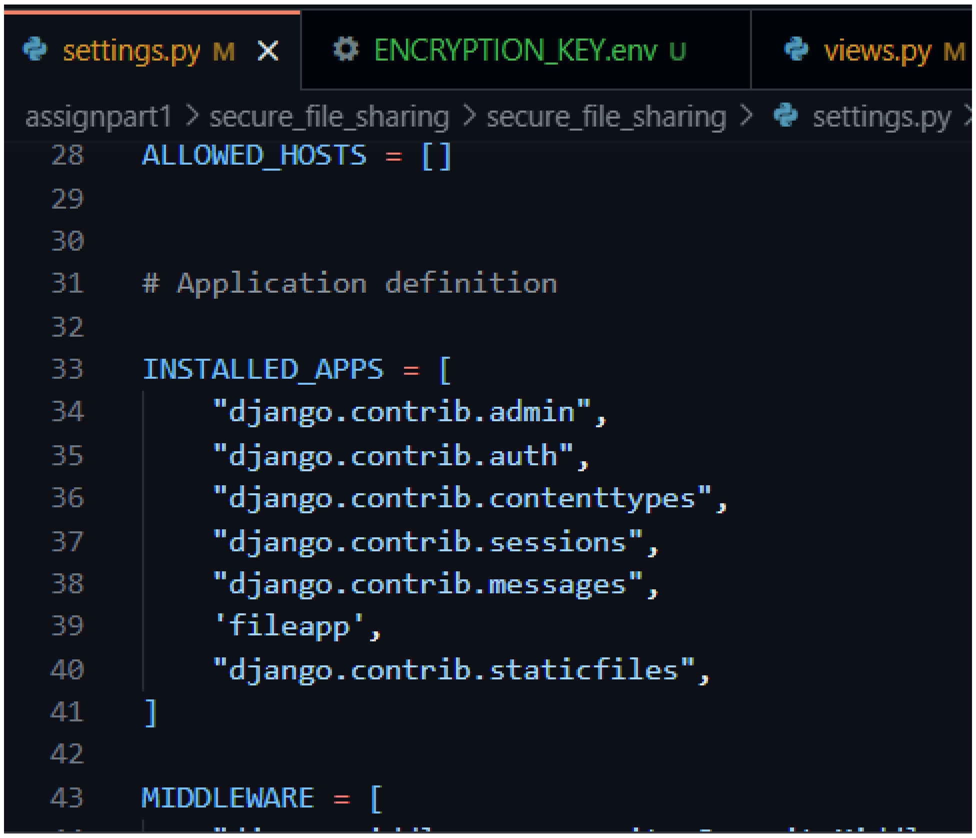

EXPLORER
ASSIGNG-GROUP14 [CODESPACE...
assignpart1
secure_file_sharing
fileapp
migrations
__init__.py
admin.py M
apps.py
models.py M
tests.py
views.py M
secure_file_sharing
__pycache__
__init__.py
asgi.py
settings.py M
urls.py M
wsgi.py
manage.py
part1reflection.md

assigng-group14 [Codespaces: effective yodel]

```
@fshajahan →/workspaces/assigng-group14/assignpart1/secure_file_sharing (main) $ python manage.py startapp fileapp
@fshajahan →/workspaces/assigng-group14/assignpart1/secure_file_sharing (main) $ 
```

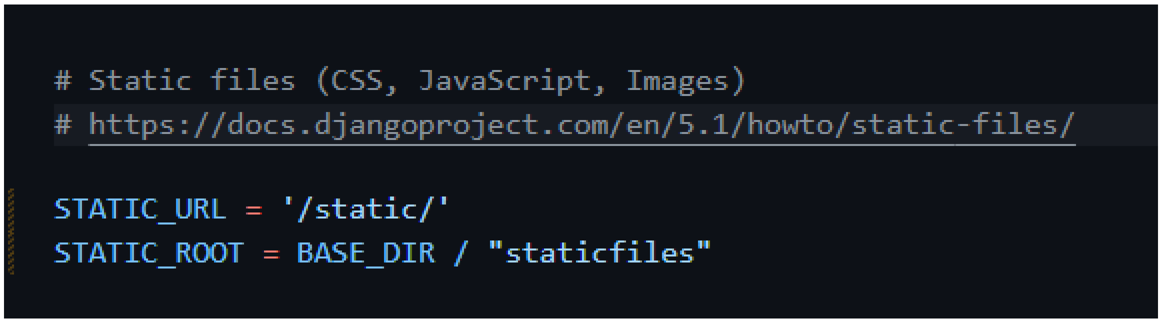
2. Configure the settings.py File:

- Added fileapp to the INSTALLED_APPS in settings.py to ensure Django knows about the new app.


settings.py M
ENCRYPTION_KEY.env U
views.py M
assignpart1 > secure_file_sharing > secure_file_sharing > settings.py

```python
ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    'fileapp',
    "django.contrib.staticfiles",
]

MIDDLEWARE = [
```

- The STATIC_URL and STATIC_ROOT settings were configured to manage static files like CSS and JavaScript for our front-end, ensuring they are served correctly during development.



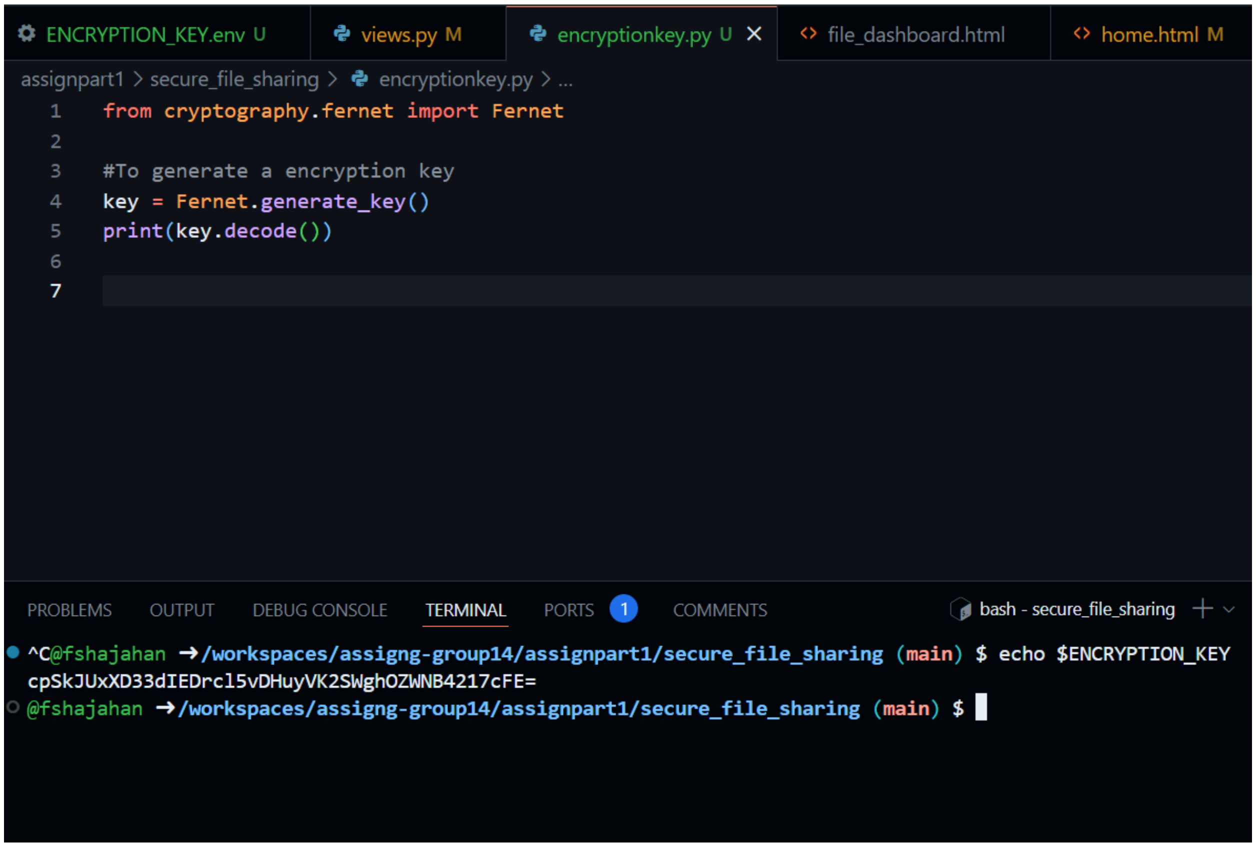
```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.1/howto/static-files/

STATIC_URL = '/static/'
STATIC_ROOT = BASE_DIR / "staticfiles"
```

3. Install Required Packages:

- We installed the necessary libraries, particularly the cryptography package (pip install cryptography), to handle file encryption and decryption. We opted for cryptography.fernet because it's simple to use, supports strong encryption, and fits our needs for symmetric encryption (same key used for encryption and decryption).
- The cryptography library is preferred because it provides a high level of security with minimal effort, and its documentation is clear and well-supported.

4. Generate and Set the Encryption Key:

- To securely encrypt files, we needed to generate a unique encryption key. We used the cryptography.fernet library to generate the key
- The code generated a new key and printed it. We then stored this key as an environment variable to avoid hardcoding it in the codebase for security reasons.



```python
from cryptography.fernet import Fernet

#To generate a encryption key
key = Fernet.generate_key()
print(key.decode())

```

```
^C@fshajahan →/workspaces/assigng-group14/assignpart1/secure_file_sharing (main) $ echo $ENCRYPTION_KEY
cpSkJUxXD33dIEDrcl5vDHuyVK2SWghOZWNB4217cFE=
@fshajahan →/workspaces/assigng-group14/assignpart1/secure_file_sharing (main) $
```

- We directly set the encryption key to the environment variable
- This ensured that the key was successfully stored in the environment and could be accessed securely during runtime.

5. Create the EncryptedFile Model:

- We created the EncryptedFile model to store the encrypted files. This model includes fields like file_name, file_content, and sha256_hash. Storing the file_content as a BinaryField ensures that encrypted files are saved in their binary form.
- The sha256_hash field is used to store a hash of the file's encrypted content. This provides an additional layer of security by allowing us to verify the integrity of the file when it is downloaded.
- We associated the model with the user who uploads the file using a ForeignKey to the User model, ensuring that only the uploader can access their files (unless shared with others).
- After defining the model, we ran python manage.py makemigrations and python manage.py migrate to create the necessary database tables.

6. Set Up File Upload and Encryption:

- In the views.py file, we created the upload_file view to handle file uploads. When a file is uploaded, we read the file's content using request.FILES['file'].
- We retrieved the encryption key from the environment variable using encryptedkey.py

    This ensured that the key was retrieved securely from the environment and used for file encryption.

- We encrypted the file content using Fernet.encrypt() and stored the encrypted content and its associated SHA-256 hash in the database.
- The choice to store files in encrypted form in the database ensures that files are secure, even if someone gains unauthorized access to the database.

7. File Download and Decryption:

- We created the download_file view to allow users to retrieve their encrypted files. When a user requests to download a file, the app fetches the file from the database, decrypts it using the same Fernet key, and serves the decrypted file for download.
- We used Fernet.decrypt() for decryption because it ensures that the file can only be decrypted with the correct key, providing security during file retrieval.
- This step ensures that users can only access their files and that the files are safely encrypted both during storage and transmission.

8. File Sharing Functionality:

- We implemented the file-sharing feature by adding a shared_with field in the EncryptedFile model, allowing users to share files with others. This field uses a ManyToManyField to associate multiple users with a single file.
- The share_file view allows users to select other users to share their files with, providing a seamless way to collaborate securely within the app.
- This ensures that only authorized users can download shared files, adding an extra layer of control over access permissions.

9. File Dashboard for Managing Files:

- We created a file dashboard page where users can see a list of all files they've uploaded or have been shared with them. The dashboard is accessed through the file_dashboard view.

- In the view, we queried the database for files associated with the logged-in user (EncryptedFile.objects.filter(user=request.user)) and files that have been shared with the user (EncryptedFile.objects.filter(shared_with=request.user)).
- This page serves as the main interface for managing uploaded and shared files.

10. Create Views for Register, Login, and Logout:

- We added views to handle user registration, login, and logout using Django's built-in authentication system. These views are accessible through the URLs /register/, /login/, and /logout/.
- We used Django's auth_views for login and logout functionalities and custom views for registration and user management, allowing us to create a smooth user experience.

11. Define URL Routing:

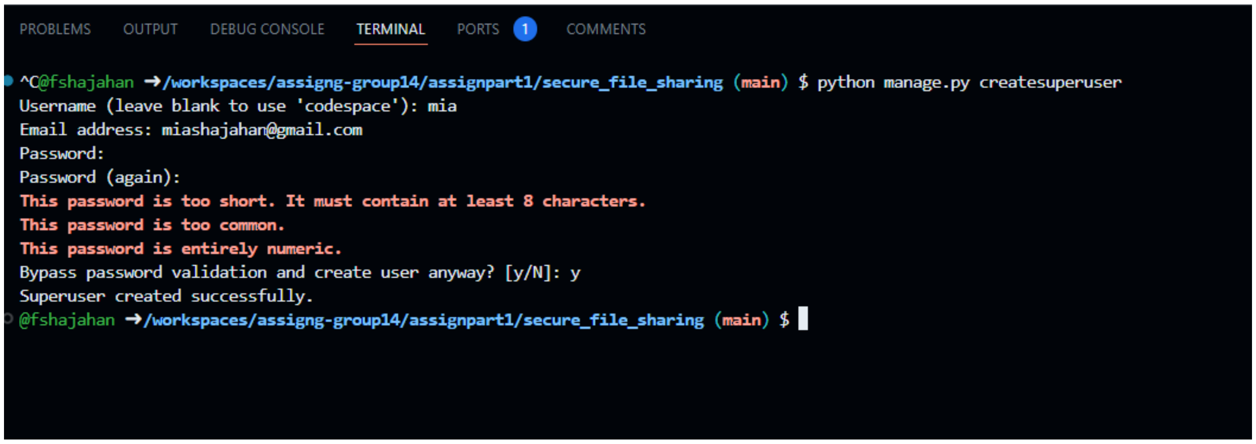
- We mapped all the views to corresponding URLs in the urls.py file. For example, the upload_file view is mapped to /upload/, and the download_file view is mapped to /download/<int:file_id>/.
- The share_file view is dynamically routed to allow users to share specific files (/share/<int:file_id>/), ensuring that each file can be uniquely identified.

12. Create HTML Templates for File Management:

- We created the necessary HTML templates to allow users to interact with the app. The templates include forms for file upload (upload.html), sharing files (share_file.html), and displaying the file dashboard (file_dashboard.html).
- These templates ensure a user-friendly interface for interacting with the file management system, making it easy for users to upload, download, share, and manage their files.

13. Create a Superuser for Admin Access:

- Before testing the application, we created a superuser account to access the Django admin panel and manage the application's data.

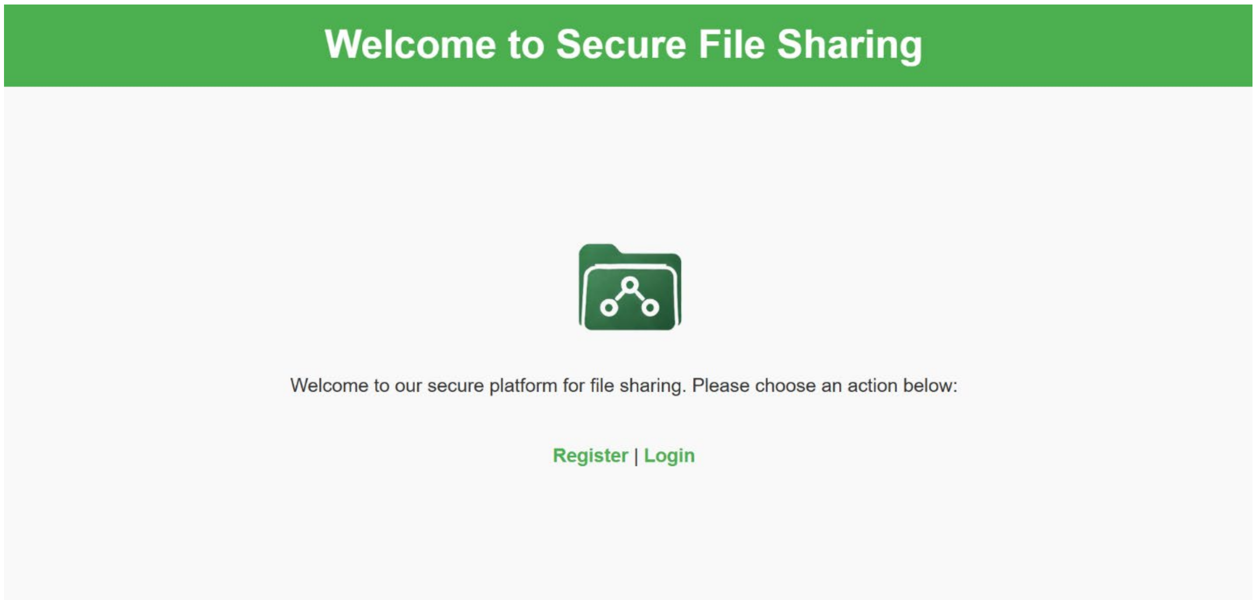


```
^C@fshajahan ➜/workspaces/assigng-group14/assignpart1/secure_file_sharing (main) $ python manage.py createsuperuser
Username (leave blank to use 'codespace'): mia
Email address: miashajahan@gmail.com
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
@fshajahan ➜/workspaces/assigng-group14/assignpart1/secure_file_sharing (main) $
```

- The command prompted us to enter a username, email, and password. The superuser account allows full administrative access to the app, enabling us to view and manage uploaded files and users through the admin interface.

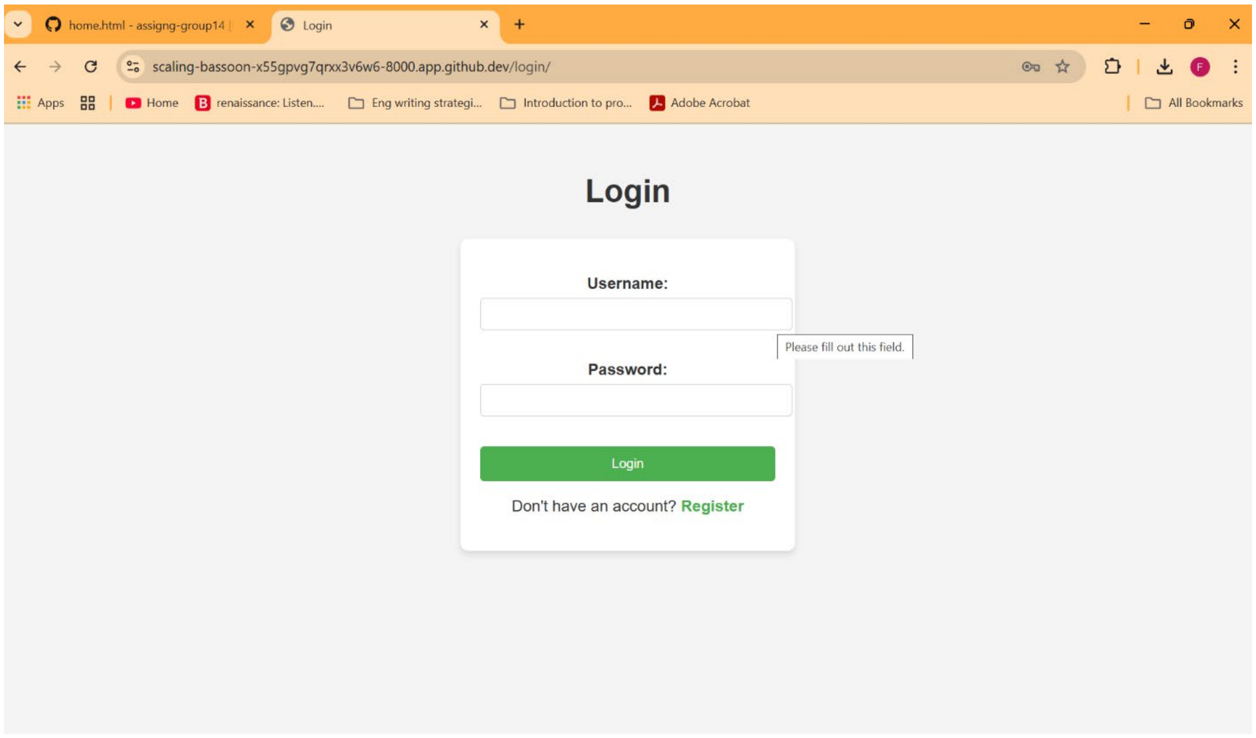
14. Test the Application:

- After setting up the superuser, we tested the app by running the Django development server using python manage.py runserver. We tested the following functionalities:
    - **Home, User Registration, Login, and Logout**: Ensured that users could register, log in, and log out properly, and that only logged-in users could upload and manage files.
    - **File Download and Decryption**: Tested downloading encrypted files and ensuring they were properly decrypted using the same encryption key.
    - **File Sharing**: Verified that files could be shared between users and that only authorized users could access shared files.
    - **File Dashboard**: Ensured that users could view a list of their files and files shared with them, and that the interface was user-friendly.
    -


Welcome to Secure File Sharing

Welcome to our secure platform for file sharing. Please choose an action below:

Register | Login



Login

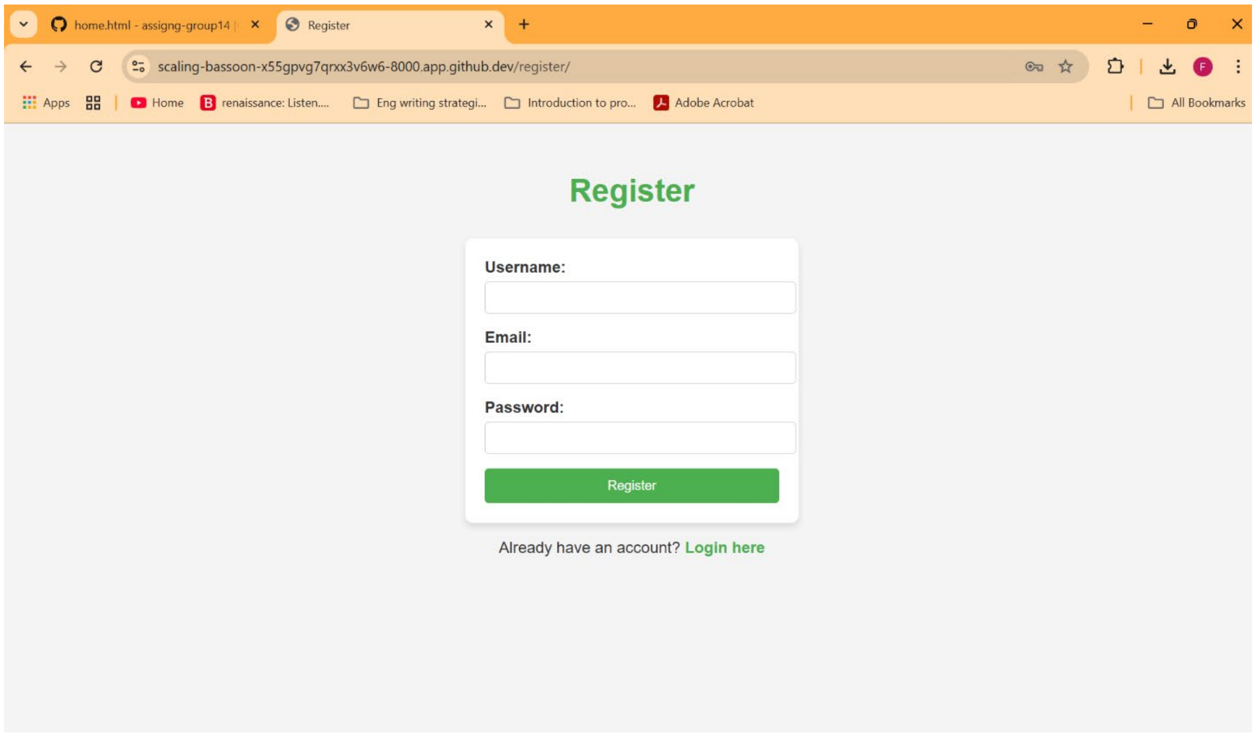
Username:

Please fill out this field.

Password:

Login

Don't have an account? Register



Register

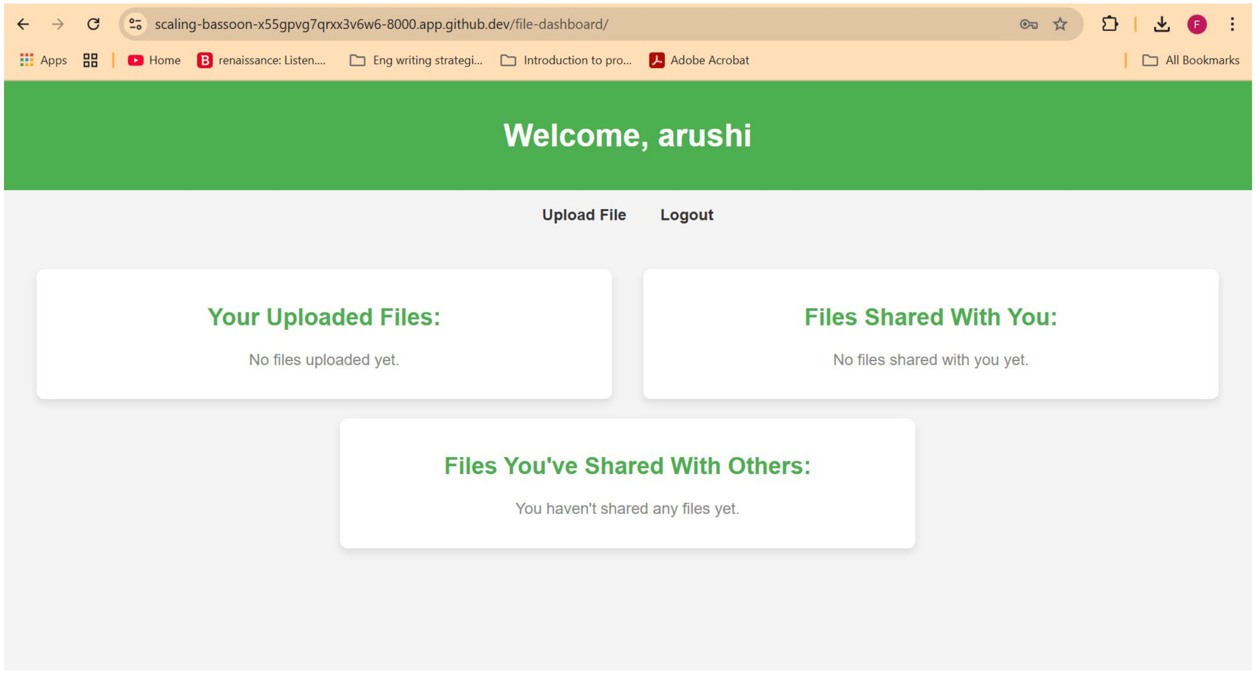
Username:

Email:

Password:

Register

Already have an account? Login here



Welcome, arushi

Upload File Logout

Your Uploaded Files:

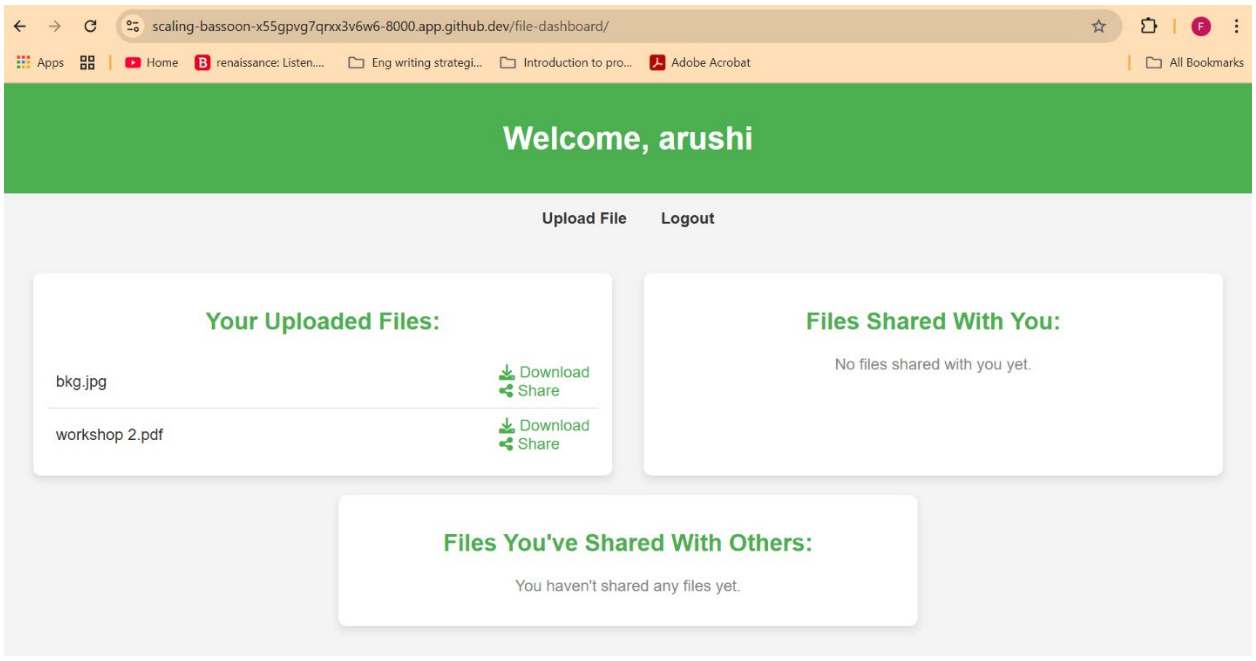
No files uploaded yet.

Files Shared With You:

No files shared with you yet.

Files You've Shared With Others:

You haven't shared any files yet.


scaling-bassoon-x55gpvg7qrxx3v6w6-8000.app.github.dev/file-dashboard/

Welcome, arushi

Upload File Logout

Your Uploaded Files:

bkg.jpg	Download Share
workshop 2.pdf	Download Share

Files Shared With You:

No files shared with you yet.

Files You've Shared With Others:

You haven't shared any files yet.

- **File Upload and Encryption**: Verified that files could be uploaded and securely encrypted, with the encryption key being retrieved from the environment variable. We confirmed that files were stored in encrypted form in the database.

- We also tested the security of the application by ensuring that files were encrypted both at rest and in transit. We confirmed that users could only download their own files or those shared with them, and that unauthorized users were not able to access files.

ALL the testing of the functionality is attached to the video records

1. We, Mia Shajahan, Arushi Gopinath declare that the attached Assignment is our own work in accordance with the Seneca Academic Policy. We have not copied any part of this Assignment, manually or electronically, from any other source, including websites, unless specified as references. We have not distributed our work to other students.

Specify what each member has done towards the completion of this work

Name	**Task(s)**
Mia Shajahan	Did the front end and backend, helped with documentation and recording
Arushi Gopinath	Collected information, helped with documentation, and did bug checking, and recording