Name: Viplavi Wade

Date: March 17, 2025

Compliance Dashboard for Tokunize

GitHub Repository Link: https://github.com/ViplaviWade/Compliance-Dashboard

1.0 Introduction

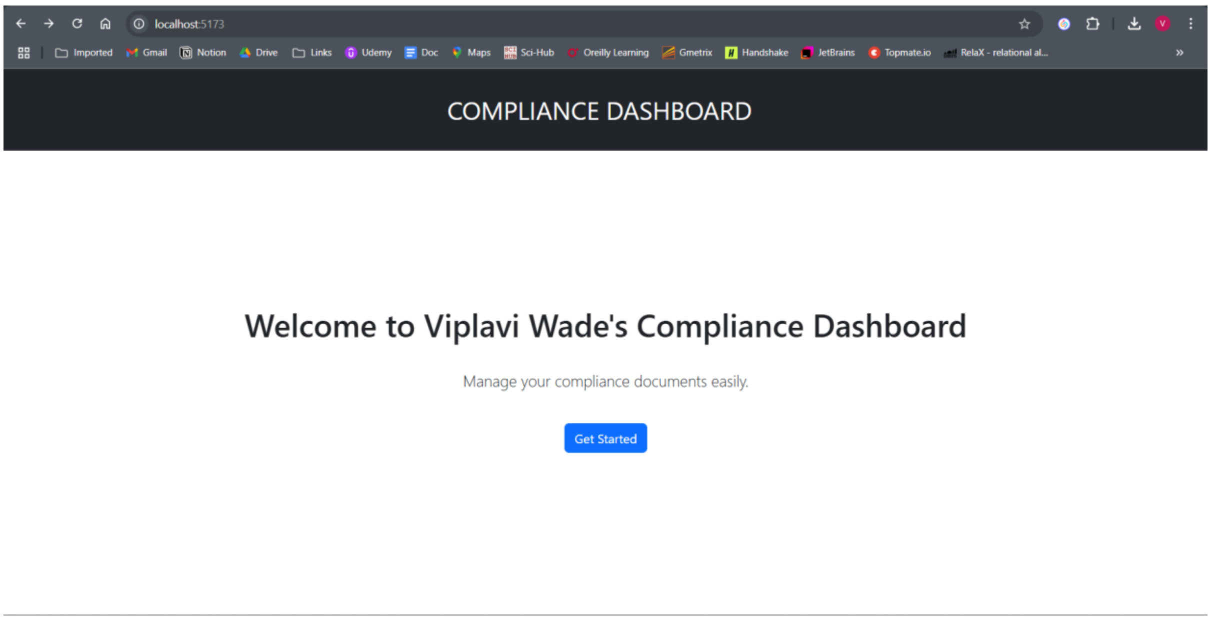
The Compliance Dashboard is designed to manage and verify KYC (Know Your Customer) and AML (Anti-Money Laundering) documents efficiently. This platform enables users to upload multiple documents while ensuring compliance with security and validation standards. The system ensures seamless document verification with advanced error handling and a user-friendly interface.

2.0 Technologies Used

The project leverages modern technologies for both frontend and backend development:

- Frontend: React.js, Bootstrap, TypeScript
- Backend: Django, Django REST Framework
- Database: PostgreSQL
- Storage: Local File System (media folder)

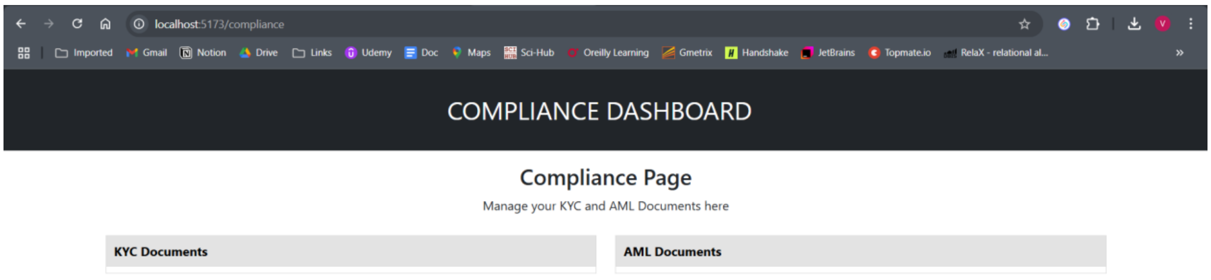
3.0 UI Design and Project flow:



Welcome to Viplavi Wade's Compliance Dashboard

Manage your compliance documents easily.

After the “Get Started” button is clicked from the main page of the application, system redirects to http://localhost:5173/compliance which is a router to this page which handles the compliance documents for KYC and AML documents. System uses two accordion cards for each type of documents. By default, the accordions are set to be closed.



COMPLIANCE DASHBOARD

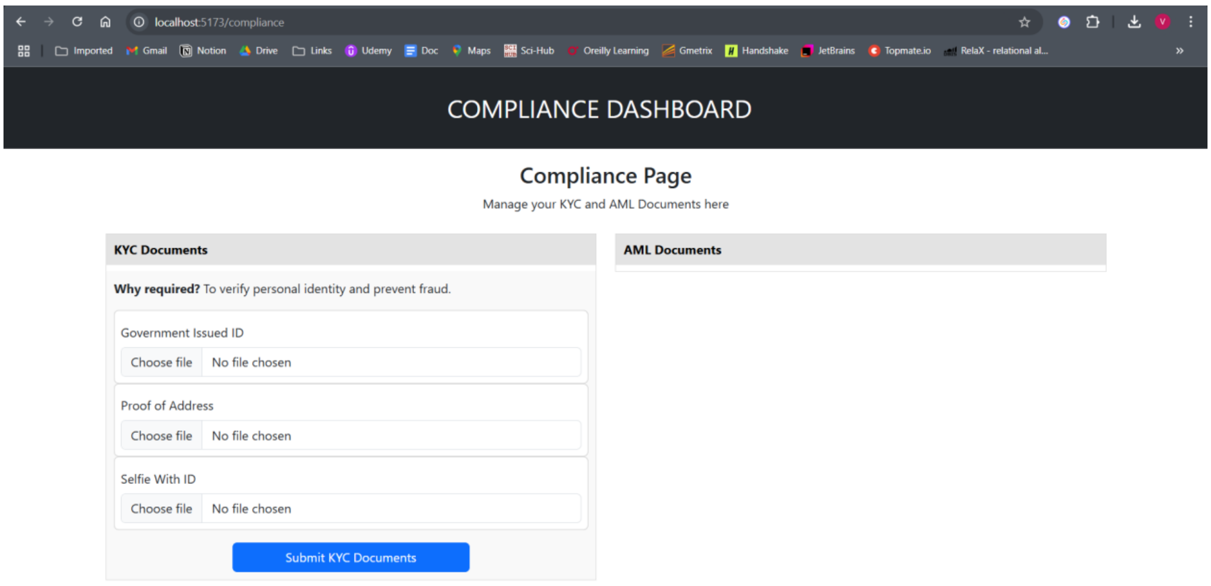
Compliance Page

Manage your KYC and AML Documents here

KYC Documents

AML Documents

When we click any of the accordion from KYC Documents and AML Documents, the accordions show up some information about respective document types, stating its use case for legal compliance.



COMPLIANCE DASHBOARD

Compliance Page

Manage your KYC and AML Documents here

KYC Documents

Why required? To verify personal identity and prevent fraud.

Government Issued ID

Choose file | No file chosen

Proof of Address

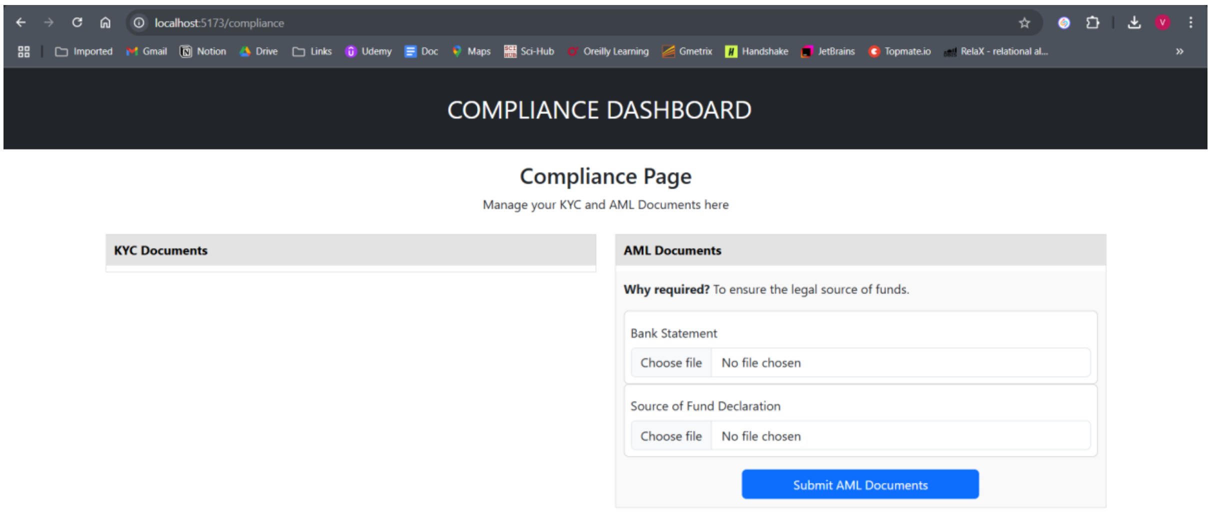
Choose file | No file chosen

Selfie With ID

Choose file | No file chosen

Submit KYC Documents

AML Documents


COMPLIANCE DASHBOARD

Compliance Page

Manage your KYC and AML Documents here

KYC Documents

AML Documents

Why required? To ensure the legal source of funds.

Bank Statement

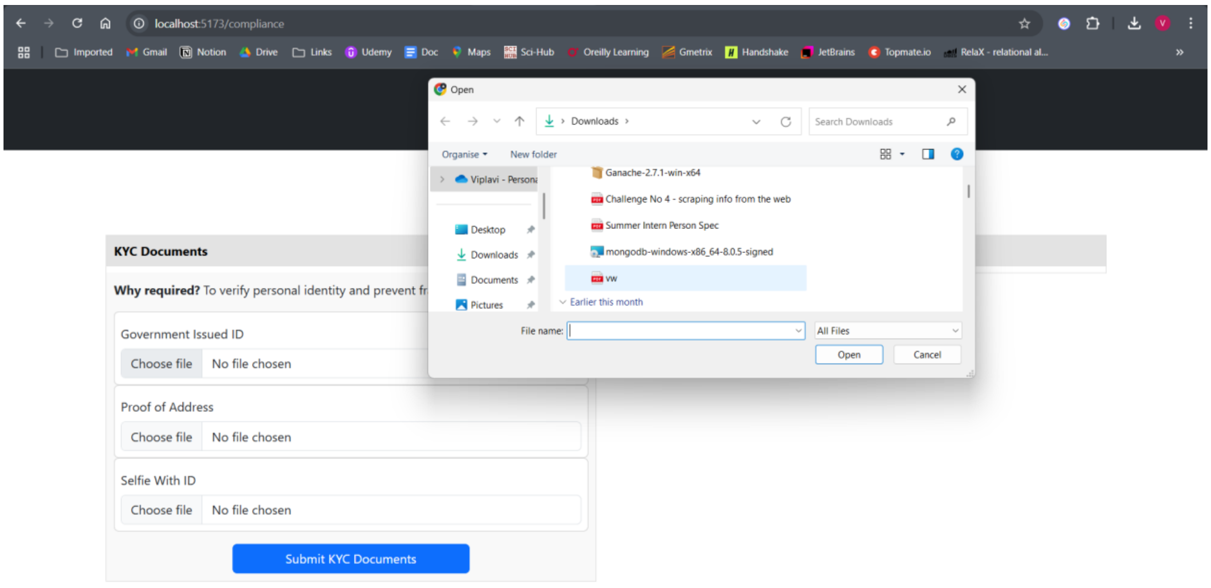
Choose file No file chosen

Source of Fund Declaration

Choose file No file chosen

Submit AML Documents

Starting with uploading a file for the given document type, after clicking the ‘Choose File’ option we can see the File Selection Dialog Box. We can choose appropriate file for our use


KYC Documents

Why required? To verify personal identity and prevent fr

Government Issued ID

Choose file No file chosen

Proof of Address

Choose file No file chosen

Selfie With ID

Choose file No file chosen

Submit KYC Documents

Open

Downloads

Ganache-2.7.1-win-x64

Challenge No 4 - scraping info from the web

Summer Intern Person Spec

mongodb-windows-x86_64-8.0.5-signed

vw

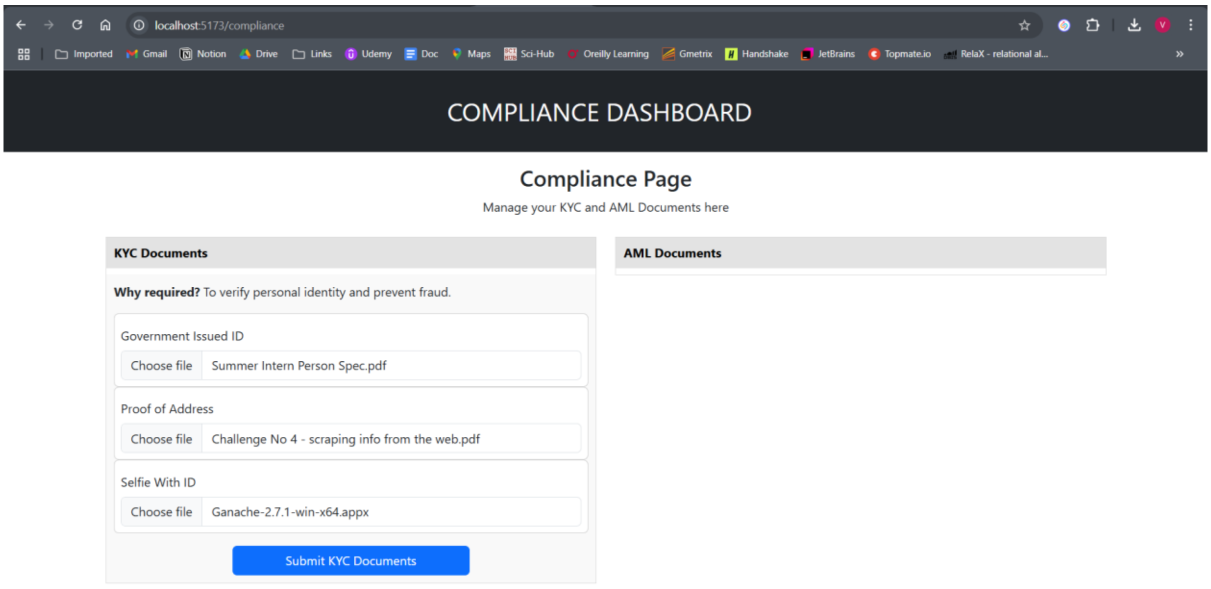
Earlier this month

File name:

All Files

Open Cancel

After selecting the files for our legal compliance, we can see the names of the document we are trying to upload.



COMPLIANCE DASHBOARD

Compliance Page

Manage your KYC and AML Documents here

KYC Documents

Why required? To verify personal identity and prevent fraud.

Government Issued ID

Choose file Summer Intern Person Spec.pdf

Proof of Address

Choose file Challenge No 4 - scraping info from the web.pdf

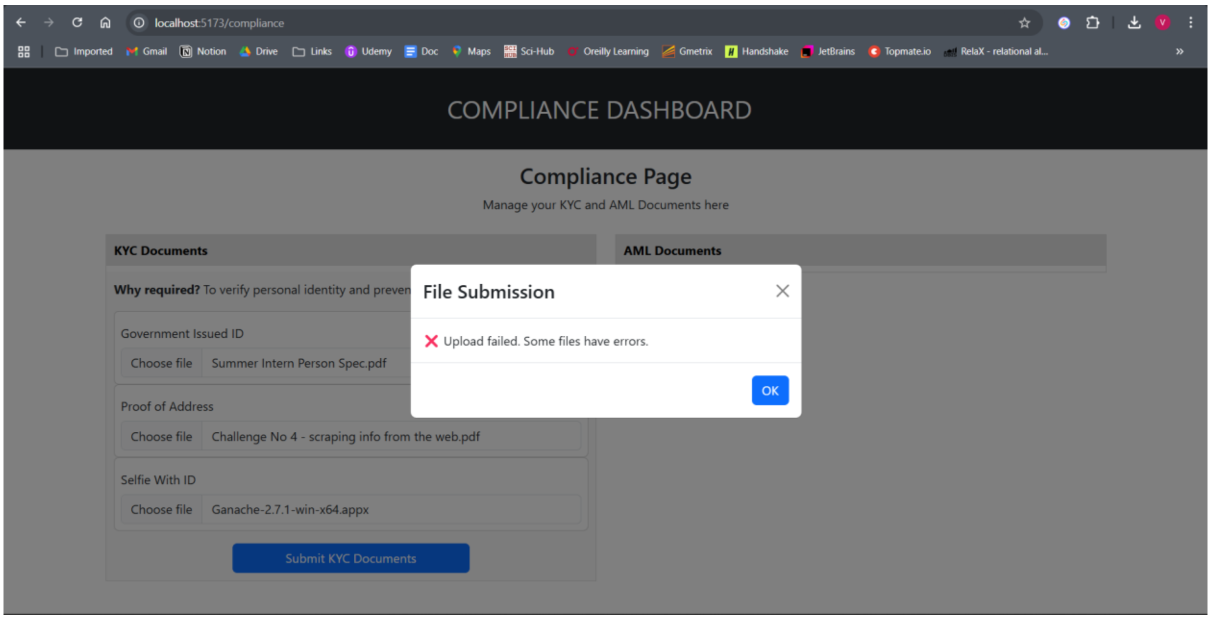
Selfie With ID

Choose file Ganache-2.7.1-win-x64.appx

Submit KYC Documents

AML Documents

In previous screenshot, we can see I uploaded a Win Zip file, which is not supported for our System, so when we click the “Submit” button the system throws an error and informs the user that the “Upload Failed”, mentioning some of the files have errors.



COMPLIANCE DASHBOARD

Compliance Page

Manage your KYC and AML Documents here

KYC Documents

Why required? To verify personal identity and preven

Government Issued ID

Choose file Summer Intern Person Spec.pdf

Proof of Address

Choose file Challenge No 4 - scraping info from the web.pdf

Selfie With ID

Choose file Ganache-2.7.1-win-x64.appx

Submit KYC Documents

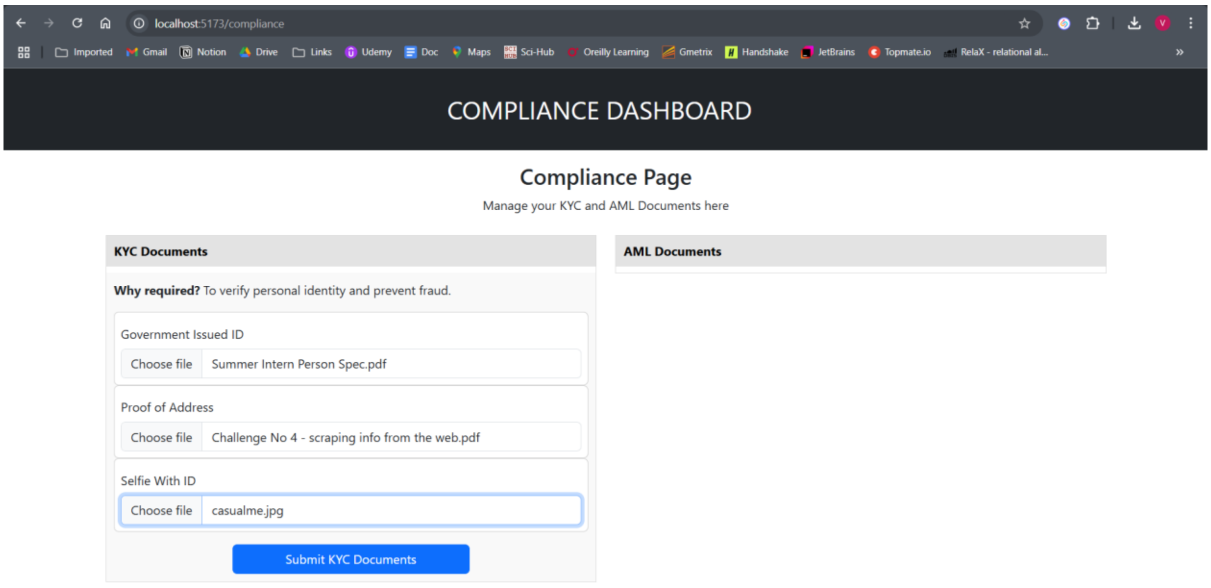
AML Documents

File Submission

❌ Upload failed. Some files have errors.

OK

To correct that, I try to upload a .jpg file which is supported by our system, in this case



COMPLIANCE DASHBOARD

Compliance Page

Manage your KYC and AML Documents here

KYC Documents

Why required? To verify personal identity and prevent fraud.

Government Issued ID
Choose file Summer Intern Person Spec.pdf

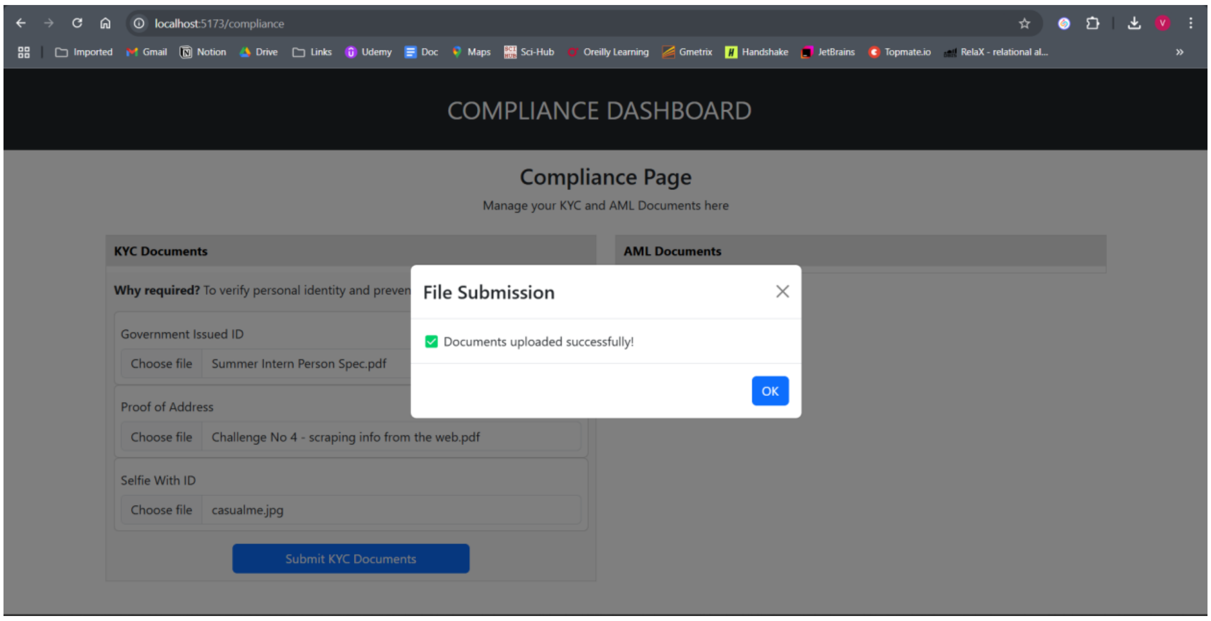
Proof of Address
Choose file Challenge No 4 - scraping info from the web.pdf

Selfie With ID
Choose file casualme.jpg

Submit KYC Documents

AML Documents

Again, when I try to submit after uploading a file which is supported by the system, we can see a Basic Modal informing the user that the Documents are successfully uploaded to the system.



COMPLIANCE DASHBOARD

Compliance Page

Manage your KYC and AML Documents here

KYC Documents

Why required? To verify personal identity and preven

Government Issued ID
Choose file Summer Intern Person Spec.pdf

Proof of Address
Choose file Challenge No 4 - scraping info from the web.pdf

Selfie With ID
Choose file casualme.jpg

Submit KYC Documents

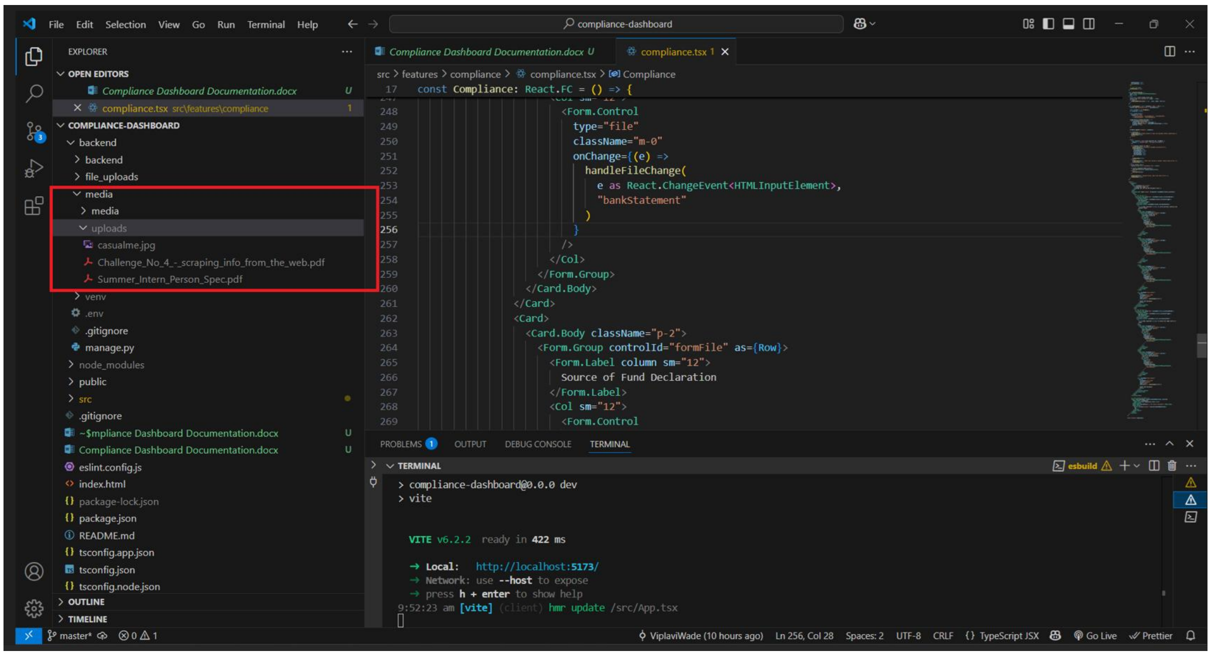
AML Documents

File Submission

✓ Documents uploaded successfully!

OK

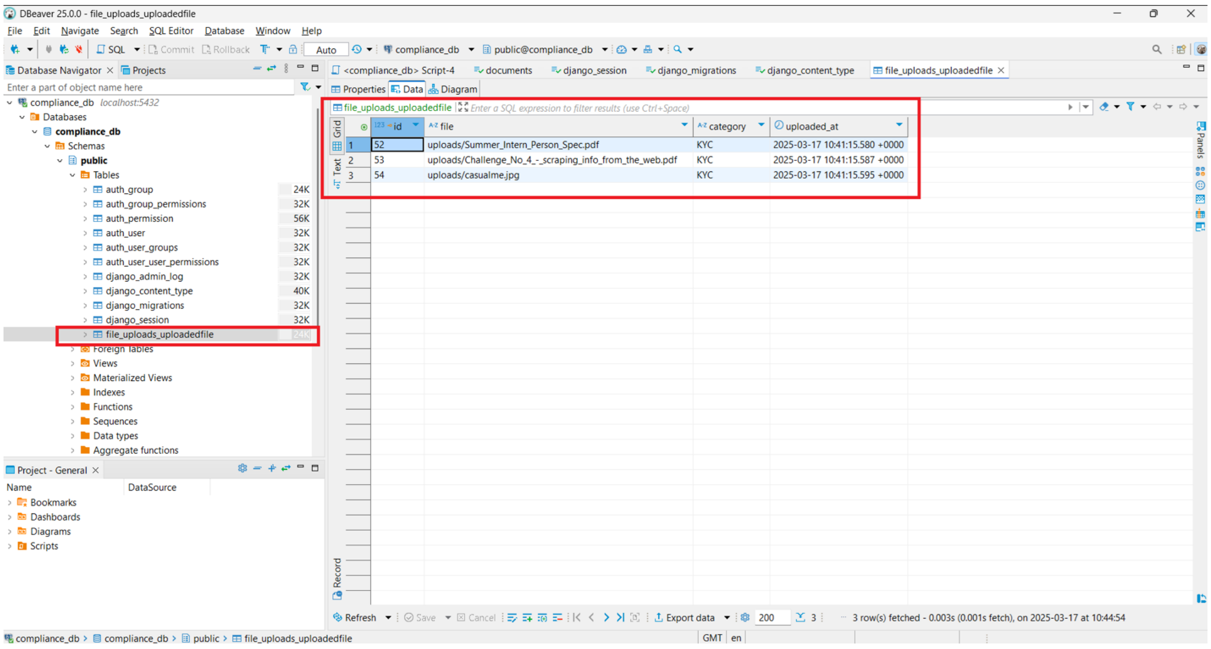
After submission the files are stored into a local storage: backend/media/uploads



```
const Compliance: React.FC = () => {
                  <Form.Control
                    type="file"
                    className="m-0"
                    onChange={(e) =>
                      handleFileChange(
                        e as React.ChangeEvent<HTMLInputElement>,
                        "bankStatement"
                      )
                    }
                  />
                </Col>
              </Form.Group>
            </Card.Body>
          </Card>
          <Card>
            <Card.Body className="p-2">
              <Form.Group controlId="formFile" as={Row}>
                <Form.Label column sm="12">
                  Source of Fund Declaration
                </Form.Label>
                <Col sm="12">
                  <Form.Control
```

```
> compliance-dashboard@0.0.0 dev
> vite


  VITE v6.2.2  ready in 422 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
9:52:23 am [vite] (client) hmr update /src/App.tsx
```

With the local storage, system also saves the file into a PostgreSQL database, inside the table 'uploadedfile'.



	id	file	category	uploaded_at
1	52	uploads/Summer_Intern_Person_Spec.pdf	KYC	2025-03-17 10:41:15.580 +0000
2	53	uploads/Challenge_No_4_-_scraping_info_from_the_web.pdf	KYC	2025-03-17 10:41:15.587 +0000
3	54	uploads/casualme.jpg	KYC	2025-03-17 10:41:15.595 +0000

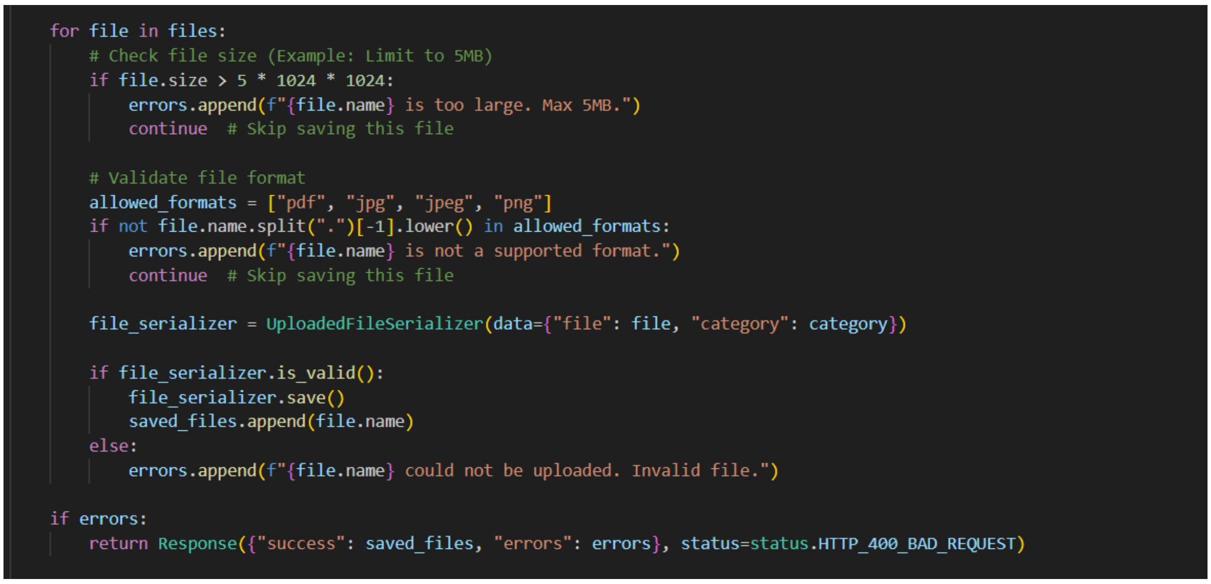
3 row(s) fetched - 0.003s (0.001s fetch), on 2025-03-17 at 10:44:54

4.0 Features Implemented

- KYC & AML Document Upload
- Multiple File Upload with Validation
- Real-time Error Handling
- React-Bootstrap UI Enhancements
- REST API for File Management

5.0 Error Handling & Validations

- File Size Restrictions (Max 5MB)
- Supported File Types (PDF, PNG, JPG)
- Invalid File Format Handling
- Corrupt File Detection



```python
for file in files:
    # Check file size (Example: Limit to 5MB)
    if file.size > 5 * 1024 * 1024:
        errors.append(f"{file.name} is too large. Max 5MB.")
        continue  # Skip saving this file

    # Validate file format
    allowed_formats = ["pdf", "jpg", "jpeg", "png"]
    if not file.name.split(".")[-1].lower() in allowed_formats:
        errors.append(f"{file.name} is not a supported format.")
        continue  # Skip saving this file

    file_serializer = UploadedFileSerializer(data={"file": file, "category": category})

    if file_serializer.is_valid():
        file_serializer.save()
        saved_files.append(file.name)
    else:
        errors.append(f"{file.name} could not be uploaded. Invalid file.")

if errors:
    return Response({"success": saved_files, "errors": errors}, status=status.HTTP_400_BAD_REQUEST)
```

6.0 Security Considerations

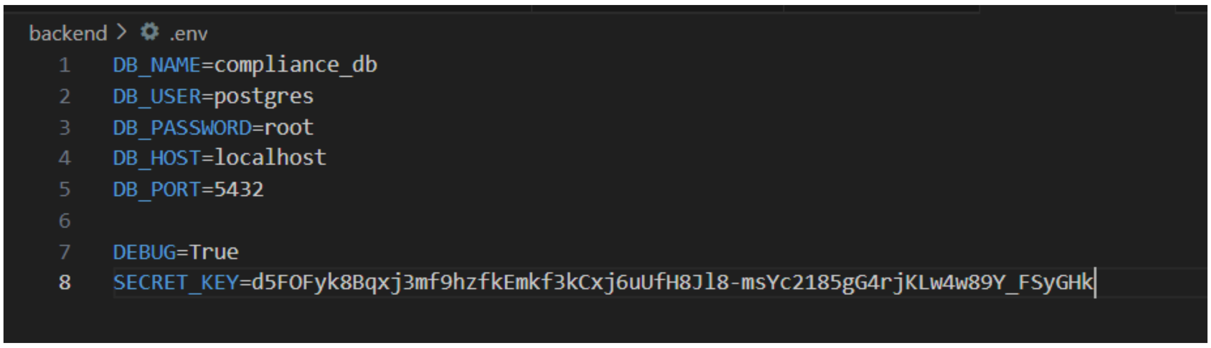
To ensure security, the system follows best practices such as:

- Environment Variables for API URLs:

I have added secured data such as:

DB_NAME,
DB_USER,
DB_PASSWORD,
DB_HOST,
DB_PORT,
SECRET_KEY

All this data has been stored in the ".env" file securely and the .env file is not committed to GitHub for security. The **SECRET_KEY** is a crucial security component in Django-based applications, including the Compliance Dashboard. While it is not explicitly referenced in the current scope of the project, it has been incorporated to future-proof the application and ensure robust security measures.
By defining and storing the **SECRET_KEY** securely in an environment variable (.env file), the project follows industry best practices, preventing accidental exposure and mitigating potential security risks.



```
DB_NAME=compliance_db
DB_USER=postgres
DB_PASSWORD=root
DB_HOST=localhost
DB_PORT=5432

DEBUG=True
SECRET_KEY=d5FOFyk8Bqxj3mf9hzfkEmkf3kCxj6uUfH8Jl8-msYc2185gG4rjKLw4w89Y_FSyGHk
```

Why is SECRET_KEY Important?

Django relies on the SECRET_KEY for various security-related tasks. Even though it may not be actively used in the current setup, it is included in the project to ensure security best practices and future scalability. Here's why it is important:

Session Management & Authentication
If user authentication is added in the future (like JWT authentication, OAuth, or API token-based access), the SECRET_KEY will be used to encrypt session data and prevent unauthorized access.

Password Hashing & Security
When users are allowed to sign in, Django uses the SECRET_KEY to hash passwords securely before storing them in the database. It ensures that even if the database is compromised, passwords cannot be easily cracked.

CSRF Protection & Signed Cookies
Django signs cookies and validates CSRF tokens using the SECRET_KEY. If CSRF protection is added later, this key will help protect against cross-site request forgery attacks.

Generating Secure Tokens
If the system implements password reset functionality, email verification, or one-time authentication tokens, Django will use the SECRET_KEY to generate and verify these tokens.

Ensuring Data Integrity
The SECRET_KEY is used internally by Django to sign and encrypt sensitive data, preventing tampering or unauthorized modifications.

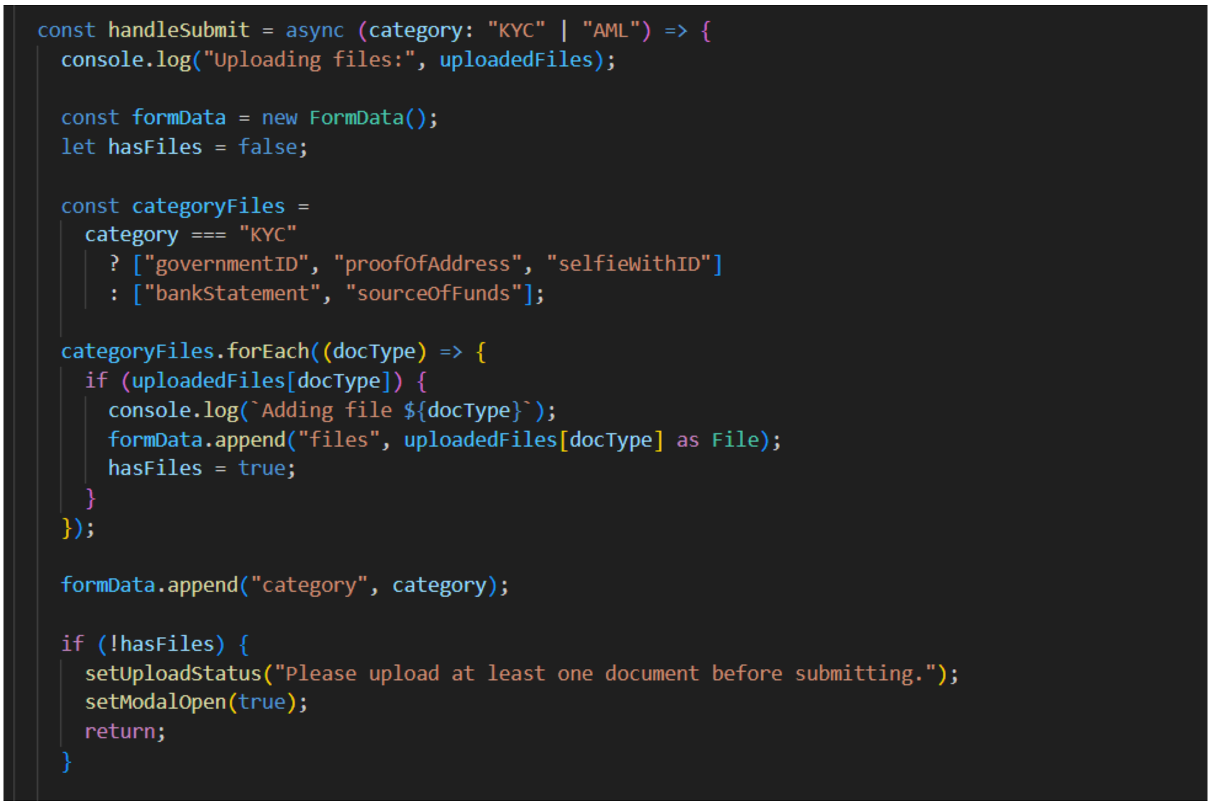
- Django File Upload Security Measures

The Compliance Dashboard ensures secure file handling by implementing various security measures to protect against malicious uploads, data breaches, and application vulnerabilities. One of the key security measures is **restricting file types**, where only PDF, JPG, PNG, and JPEG formats are allowed, preventing execution of malicious scripts that could be embedded in files. Additionally, a **file size limit of 5MB** is enforced to prevent Denial-of-Service (DoS) attacks and ensure efficient storage management. This restriction ensures that users cannot upload excessively large files that could crash the server or consume unnecessary resources.

- SQL Injection

SQL Injection is a well-known security risk where attackers try to manipulate database queries by injecting malicious SQL code. To prevent this, the Compliance Dashboard is designed with multiple security measures. Instead of writing raw SQL queries, we use Django ORM (Object-Relational Mapping), which automatically handles user inputs safely by escaping and sanitizing them. This means that even if someone tries to inject harmful SQL code, it won't execute because Django ensures that all inputs are treated as data, not as executable commands.

Another important security measure in our system is strict input validation. Users can only upload files under predefined categories like "KYC" and "AML", and the system does not allow any arbitrary values. This restriction makes sure that attackers cannot insert SQL commands into input fields.

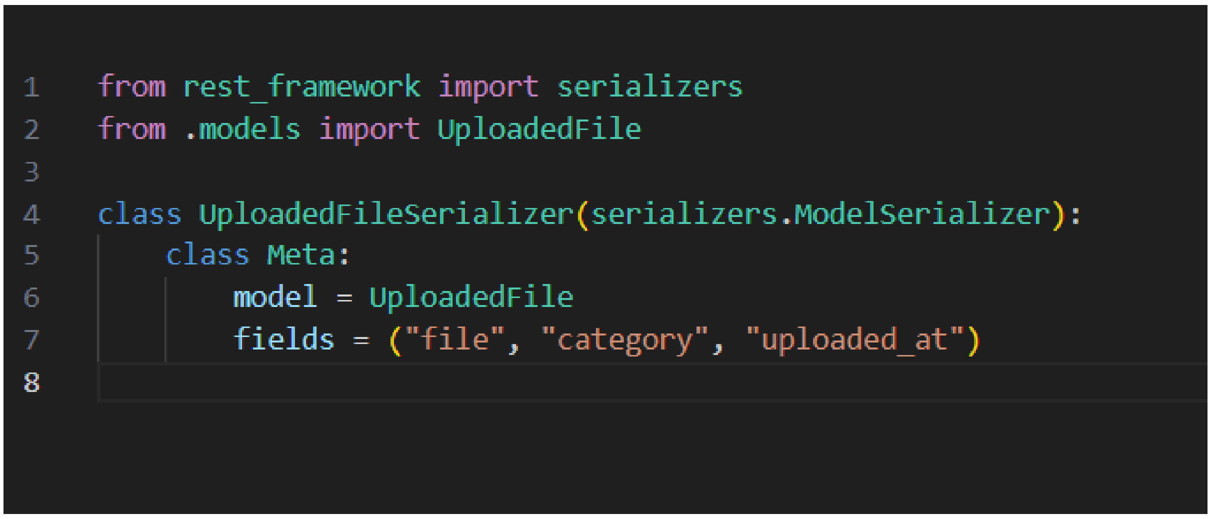


```
const handleSubmit = async (category: "KYC" | "AML") => {
  console.log("Uploading files:", uploadedFiles);

  const formData = new FormData();
  let hasFiles = false;

  const categoryFiles =
    category === "KYC"
      ? ["governmentID", "proofOfAddress", "selfieWithID"]
      : ["bankStatement", "sourceOfFunds"];

  categoryFiles.forEach((docType) => {
    if (uploadedFiles[docType]) {
      console.log(`Adding file ${docType}`);
      formData.append("files", uploadedFiles[docType] as File);
      hasFiles = true;
    }
  });

  formData.append("category", category);

  if (!hasFiles) {
    setUploadStatus("Please upload at least one document before submitting.");
    setModalOpen(true);
    return;
  }
```

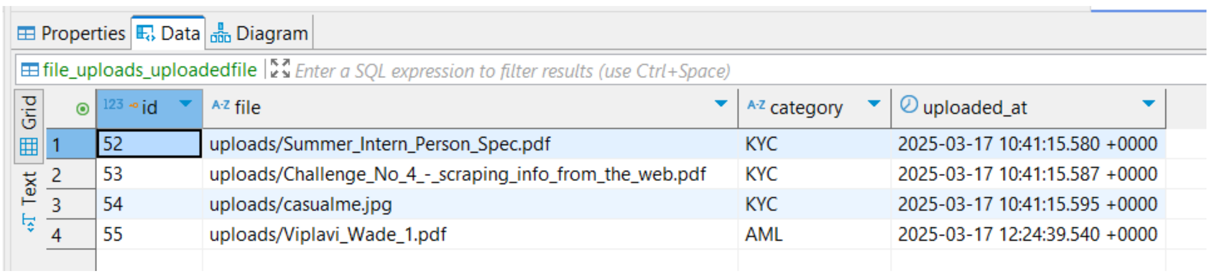
Django also parameterizes queries by default, meaning it automatically ensures that user inputs are safely handled before sending them to the database. This prevents any risk of direct SQL code execution from user input. By using Django's built-in protections and best practices, the Compliance Dashboard significantly reduces the chances of SQL Injection and keeps the database secure.

7.0 Database Schema

The system uses PostgreSQL to store document metadata. The table structure is:
Table: file_uploads_uploadedfile
Columns:
- id (Primary Key)
- file (File Path)
- category (KYC/AML)
- uploaded_at (Timestamp)



```python
from rest_framework import serializers
from .models import UploadedFile

class UploadedFileSerializer(serializers.ModelSerializer):
    class Meta:
        model = UploadedFile
        fields = ("file", "category", "uploaded_at")

```


Properties
Data
Diagram
file_uploads_uploadedfile
Enter a SQL expression to filter results (use Ctrl+Space)
Grid
Text

	id	file	category	uploaded_at
1	52	uploads/Summer_Intern_Person_Spec.pdf	KYC	2025-03-17 10:41:15.580 +0000
2	53	uploads/Challenge_No_4_-_scraping_info_from_the_web.pdf	KYC	2025-03-17 10:41:15.587 +0000
3	54	uploads/casualme.jpg	KYC	2025-03-17 10:41:15.595 +0000
4	55	uploads/Viplavi_Wade_1.pdf	AML	2025-03-17 12:24:39.540 +0000

8.0 Enhancements & Future Improvements
Future upgrades to enhance functionality include:

- AWS S3 Storage Integration
- Bulk File Upload Feature
- User Authentication for Secure Uploads