Bahir Dar University

Bahir Dar institute of technology

Department of Cyber Security

Secure note application with Django

Name

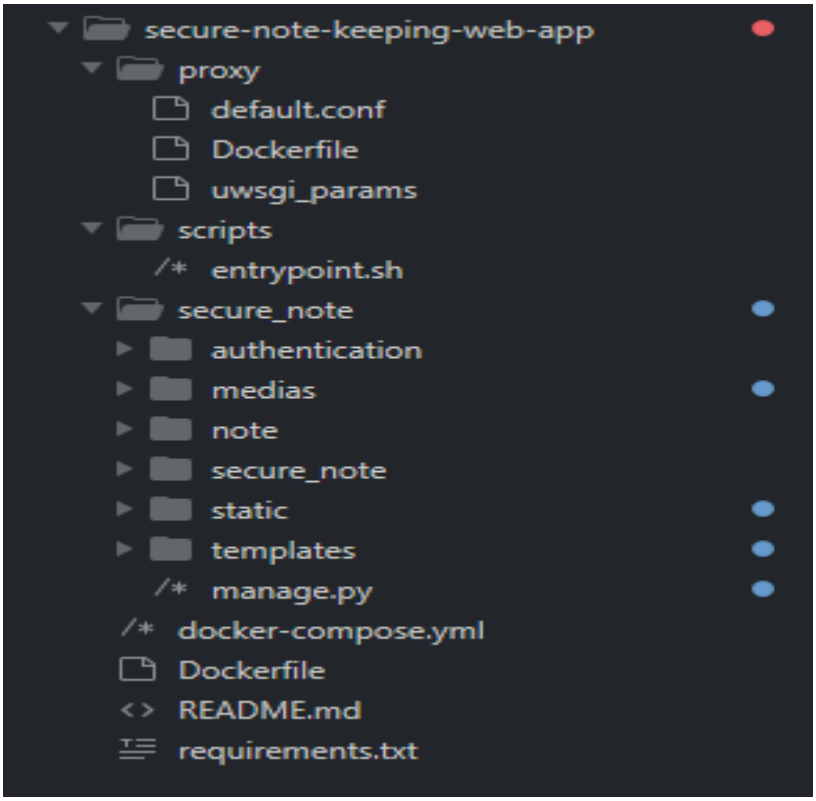
1. Mintesnot afework
2. Bisahw tesema
3. Mikiyas sisay

Submitted to: Mr. hailu

Table of Contents

Introduction

In this assignment we tried to make a secure note using html, CSS and JavaScript as a frontend, Django as a backend, nginx as a web server and Docker a deployment tool for containerizing our web application. As a starting point we want to show the high level over view of our application and parts that contain basic and security related parts.



The first folder we get is the proxy folder which contain the basic configuration of our web server which is nginx. As we can see it contain three parts which are the Dockerfile, default.conf and uwsgi-params. We will get deep in the detailed section but the main purpose of this part is accepting user http and https requests at port 80 and 443 and forward them to the Django app or fetch static file to server based on the default.conf configuration.

The second folder is the scripts folder which is the folder that contain the entry script to our application. It perform basic Django application based things like collecting static files and making migration to the database.

The third folder is the secure_note which is the main Django application that contain the Django configuration, path routing, and authentication and more codes for our application. It can be also called the backend mostly which means the one that perform the logic behind. We will get in deep dive by looking at the security, forms and models later but for now it is enough as introduction.

The two before the last files are the one that contain our Docker configuration and guide for the Docker engine on how to build this app. We will see their security implication in the last section. But for now the Docker file is for the python based image and the Docker-compose guide for building the whole app.

The last file is the one that contain our dependency libraries but it is required from to anything with it the Docker image building process will take care of it.

Installation guide

1. Requirements

Before the installation there are things that are needed to be fulfilled before doing anything with this web application.

- The system must be Linux to run the following app(Preferred if it is a Delian 12 or bookworm) Because we have containerized the system using a light weight image called alpine Linux and for Docker it is important to have the same kernel as the image under to run it.
- There must a pre-configured Docker engine that has been installed from the Docker official. (Caution: it is not recommended to use docker.io and other as a Docker engine). May be to configure Docker here is a YouTube link to Chris YouTube channel **(https://www.youtube.com/watch?v=94VQvRpjfO8&t=726s)** and the link for Docker configuration guide from Docker developers **(https://docs.docker.com/engine/install/).**
- 200 MG free space

2. Installation

If the above requirements are fulfilled we can proceed to the installation process which is very easy to do so.

1. We have used achieve with zip to secure the app. Before extracting it make sure that the one that we have sent is the one that you receive by computing the md5 hash of the tar and compare it with **md5hash**, if it match proceed to the next step. If not contact bishaw please.

```
12:54:46 ~/python/django/secure-note-keeping-web-app $$ docker pull postgres:alpine
12:54:59 ~/python/django/secure-note-keeping-web-app $$ docker pull nginxinc/nginx-unprivileged:1-alpine
12:55:04 ~/python/django/secure-note-keeping-web-app $$ docker pull python:3-alpine
```

2. To optimize our time effectively it is recommended to do this step but it is not mandatory we can skip it. This step will make a pull from the docker hub to get every required image for this assignment

```
10:11:09 ~/Desktop/pyt $$ /bin/ls
securenote.tar.xz
10:11:10 ~/Desktop/pyt $$ xz -cdv securenote.tar.xz > securenote.tar
securenote.tar.xz (1/1)
  100 %        46.5 MiB / 129.3 MiB = 0.360                    0:02
10:11:36 ~/Desktop/pyt $$ /bin/ls
securenote.tar  securenote.tar.xz
10:11:42 ~/Desktop/pyt $$ tar -xf securenote.tar
10:11:57 ~/Desktop/pyt $$ /bin/ls
secure-note-keeping-web-app  securenote.tar  securenote.tar.xz
10:12:00 ~/Desktop/pyt $$ cd secure-note-keeping-web-app
10:12:10 ~/Desktop/pyt/secure-note-keeping-web-app $$ /bin/ls
docker-compose.yml  proxy      requirements.txt  secure_note
Dockerfile          README.md  scripts
```

3. Then run the above command to extract

```
$$ docker compose up --build
```

4. Then change directory to the extracted folder and run the docker compose up --build as show in the above.

```
10:12:17 ~/Desktop/pyt/secure-note-keeping-web-app $$ docker compose up
[+] Running 1/4
 ✔ Network secure-note-keeping-web-app_default          Created
 ⠏ Container pgdb                                      Created
 ⠏ Container secure-note-keeping-web-app-secure_note-1  Created
 ⠏ Container secure-note-keeping-web-app-proxy-1        Created
Attaching to pgdb, proxy-1, secure_note-1
```

```
secure_note-1  |     errors.extend(model.check(**kwargs))
secure_note-1  |                   ^^^^^^^^^^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/db/models/base.py", line 1610, in check
secure_note-1  |     *cls._check_constraints(databases),
secure_note-1  |      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/db/models/base.py", line 2460, in _check_constraints
secure_note-1  |     connection.features.supports_nulls_distinct_unique_constraints
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/utils/functional.py", line 47, in __get__
secure_note-1  |     res = instance.__dict__[self.name] = self.func(instance)
secure_note-1  |                                          ^^^^^^^^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/db/backends/postgresql/features.py", line 141, in is_postgresql_15
secure_note-1  |     return self.connection.pg_version >= 150000
secure_note-1  |            ^^^^^^^^^^^^^^^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/utils/functional.py", line 47, in __get__
secure_note-1  |     res = instance.__dict__[self.name] = self.func(instance)
secure_note-1  |                                          ^^^^^^^^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/db/backends/postgresql/base.py", line 438, in pg_version
secure_note-1  |     with self.temporary_connection():
secure_note-1  |   File "/usr/local/lib/python3.12/contextlib.py", line 137, in __enter__
secure_note-1  |     return next(self.gen)
secure_note-1  |            ^^^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/db/backends/base/base.py", line 691, in temporary_connection
secure_note-1  |     with self.cursor() as cursor:
secure_note-1  |          ^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/utils/asyncio.py", line 26, in inner
secure_note-1  |     return func(*args, **kwargs)
secure_note-1  |            ^^^^^^^^^^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/db/backends/base/base.py", line 316, in cursor
secure_note-1  |     return self._cursor()
secure_note-1  |            ^^^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/db/backends/base/base.py", line 292, in _cursor
secure_note-1  |     self.ensure_connection()
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/utils/asyncio.py", line 26, in inner
secure_note-1  |     return func(*args, **kwargs)
secure_note-1  |            ^^^^^^^^^^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/db/backends/base/base.py", line 274, in ensure_connection
secure_note-1  |     with self.wrap_database_errors:
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/db/utils.py", line 91, in __exit__
secure_note-1  |     raise dj_exc_value.with_traceback(traceback) from exc_value
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/db/backends/base/base.py", line 275, in ensure_connection
secure_note-1  |     self.connect()
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/utils/asyncio.py", line 26, in inner
secure_note-1  |     return func(*args, **kwargs)
secure_note-1  |            ^^^^^^^^^^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/db/backends/base/base.py", line 256, in connect
secure_note-1  |     self.connection = self.get_new_connection(conn_params)
secure_note-1  |                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/utils/asyncio.py", line 26, in inner
secure_note-1  |     return func(*args, **kwargs)
secure_note-1  |            ^^^^^^^^^^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/django/db/backends/postgresql/base.py", line 277, in get_new_connection
secure_note-1  |     connection = self.Database.connect(**conn_params)
secure_note-1  |                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
secure_note-1  |   File "/usr/local/lib/python3.12/site-packages/psycopg2/__init__.py", line 122, in connect
secure_note-1  |     conn = _connect(dsn, connection_factory=connection_factory, **kwasync)
secure_note-1  |            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
secure_note-1  | django.db.utils.OperationalError: connection to server at "pgdb" (172.18.0.2), port 5432 failed: Connection refused
secure_note-1  |       Is the server running on that host and accepting TCP/IP connections?
secure_note-1  |
```

```
pgdb           | 2024-04-27 09:59:37.550 UTC [36] LOG:  database system is shut down
pgdb           |  done
pgdb           | server stopped
pgdb           |
pgdb           | PostgreSQL init process complete; ready for start up.
pgdb           |
pgdb           | 2024-04-27 09:59:37.608 UTC [1] LOG:  starting PostgreSQL 16.2 on x86_64-pc-linux-musl, compiled by gcc (Alpi
ne 13.2.1_git20231014) 13.2.1 20231014, 64-bit
pgdb           | 2024-04-27 09:59:37.608 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
pgdb           | 2024-04-27 09:59:37.608 UTC [1] LOG:  listening on IPv6 address "::", port 5432
pgdb           | 2024-04-27 09:59:37.620 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
pgdb           | 2024-04-27 09:59:37.641 UTC [50] LOG:  database system was shut down at 2024-04-27 09:59:37 UTC
pgdb           | 2024-04-27 09:59:37.656 UTC [1] LOG:  database system is ready to accept connections
```

5. If it generate error like that it is because Django tries to migrate before instating the Postgres database so wait for few second until you see the above message Ctrl + C to exit and run docker compose up.

```
10:24:07 ~/Desktop/pyt/secure-note-keeping-web-app $$ ls
total 16K
-rw-r--r-- 1 mintesnot mintesnot 647 Apr 25 13:57 docker-compose.yml
-rw-r--r-- 1 mintesnot mintesnot 596 Apr 24 22:02 Dockerfile
drwxr-xr-x 1 mintesnot mintesnot  68 Apr 24 20:20 proxy
-rw-r--r-- 1 mintesnot mintesnot 147 Apr 24 20:12 README.md
-rw-r--r-- 1 mintesnot mintesnot 141 Apr 24 20:20 requirements.txt
drwxr-xr-x 1 mintesnot mintesnot  26 Apr 24 20:20 scripts
drwxr-xr-x 1 mintesnot mintesnot 118 Apr 24 20:20 secure_note
10:24:08 ~/Desktop/pyt/secure-note-keeping-web-app $$ docker build .
```

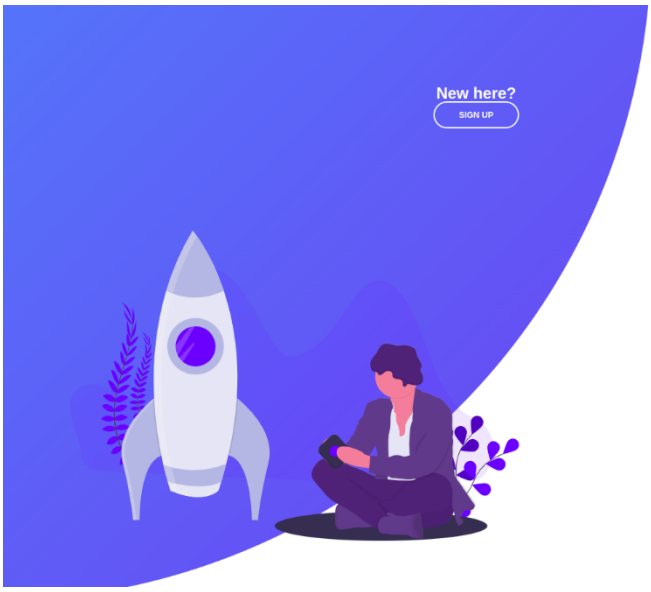
6. Build if like to build the image first and run the app use the above option run using the set 4 option.

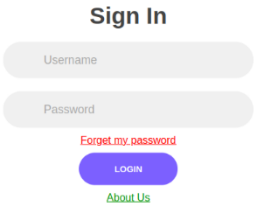
```
settings.py
26
27  ALLOWED_HOSTS = ["*"]
28  CSRF_TRUSTED_ORIGINS = ["https://localhost","https://127.0.0.1","https://mintesnot.afework"]
29
```

7. After that if you want to use the mintesnot.afework domain edit the /etc/hosts file and add the above entry(Due to security rules found in the setting file it is not allowed to other domain to access the page more specifically it will not accept any form from you if do not use either of localhost, 127.0.0.1 and minesnot.afework)
8. Get to your preferred web browser and look for https://localhost or https://127.0.0.1 or https://mintesnotafework(if you add the entry to the /etc/hosts file)(**caution** you can use either of http or https if you use http, you will get a permanent redirect to the https page).
9. Congratulations you have finished the first phase.

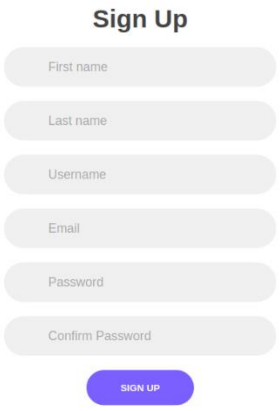
How to interact with

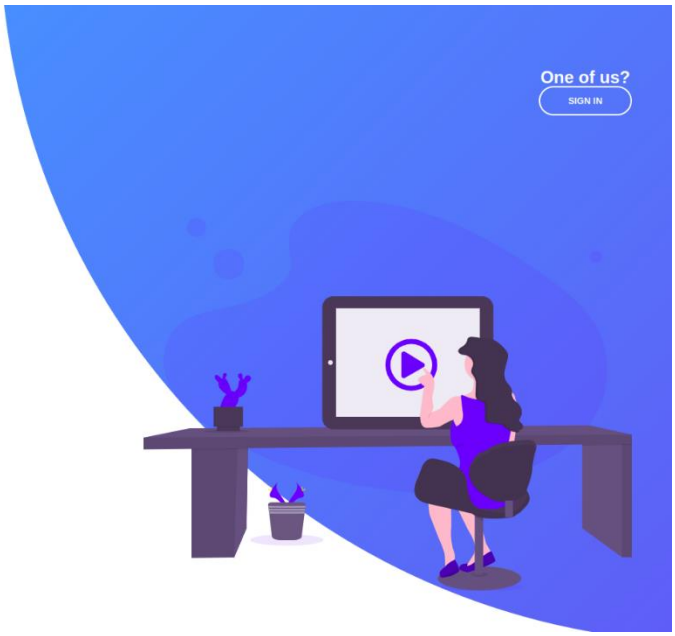
1. Login page: is the page used to accept the registered username and password to create them a live session by authenticating them.



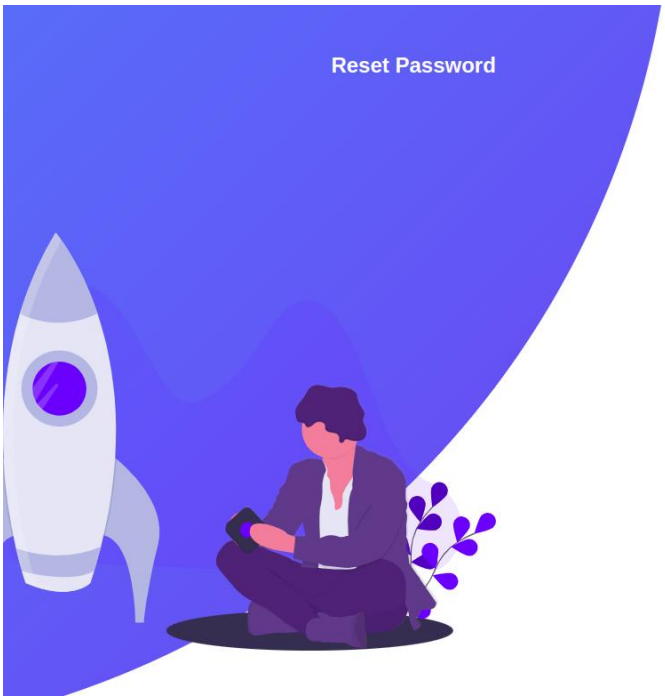


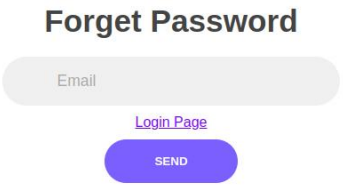
2. Registration Page: is the page used to add new user to our system



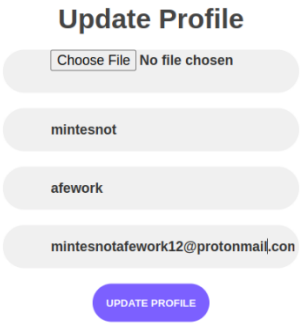


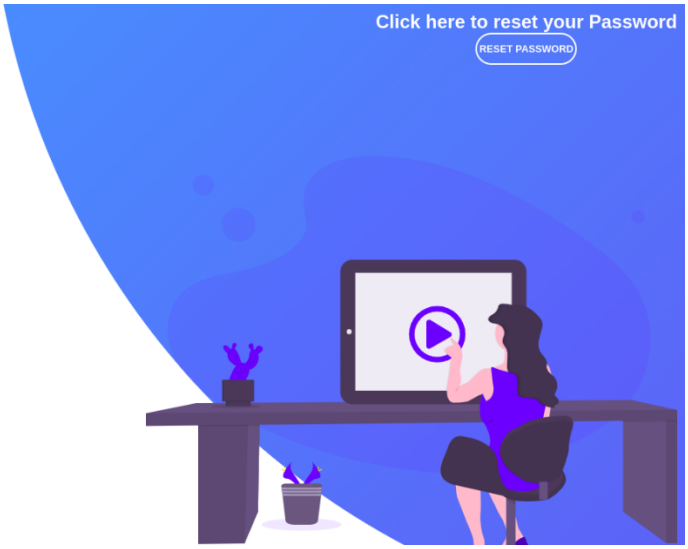
3. Forget Page: is the page used to reset password using email



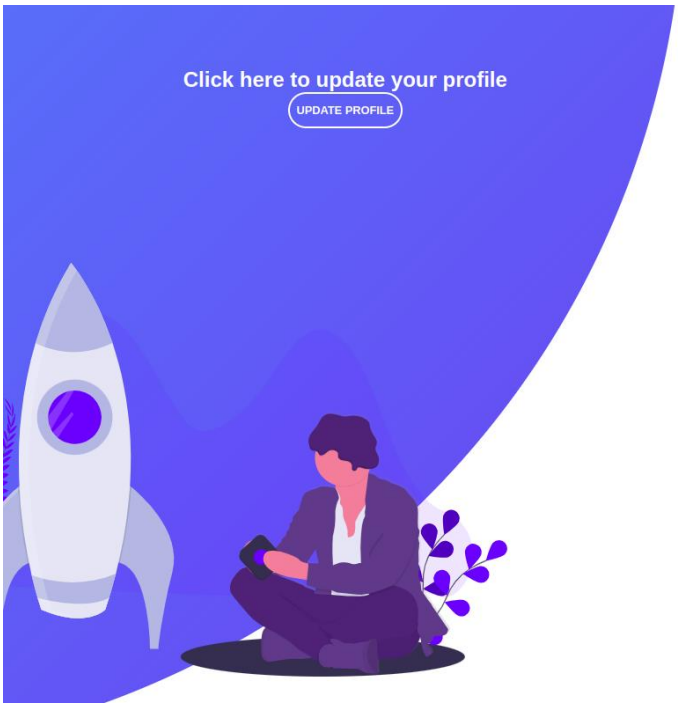


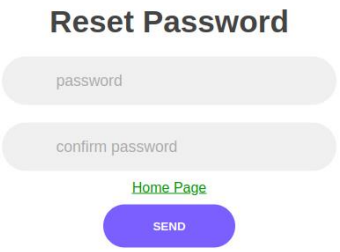
4. Update Page: is used to update user profile picture, first name, last name and email



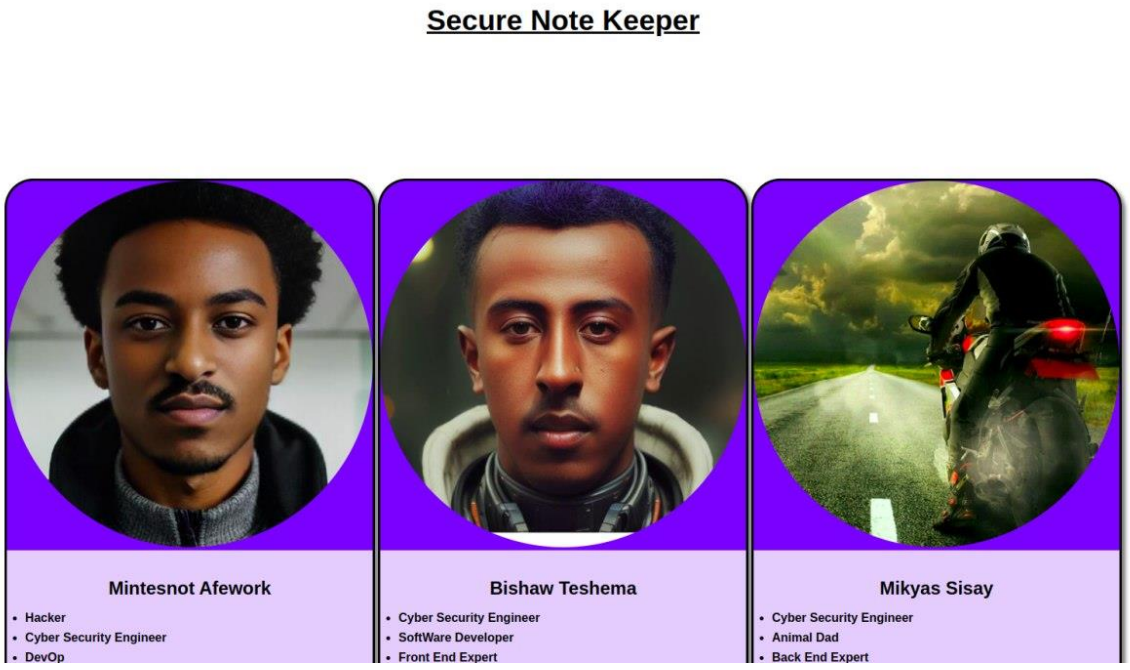


5. Reset page: is used to change our password





6. About Us Page: the page that has information about the creates and developer of this project

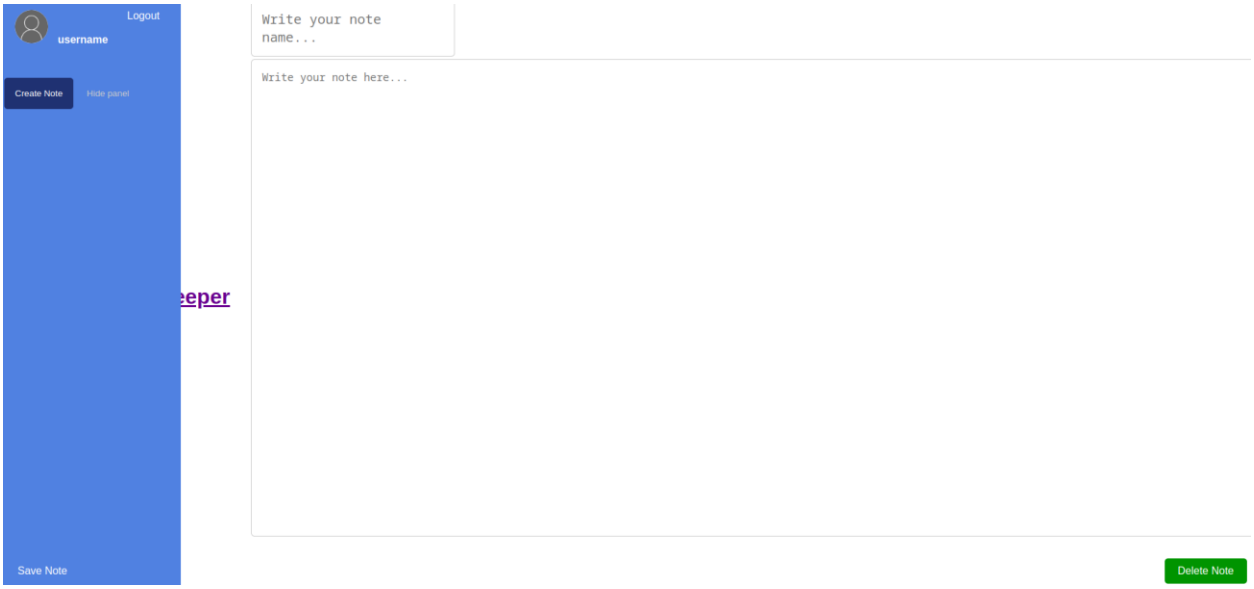


7. Home Page: is the one that used for creation new file, delete the existing file, edit it and view it with the hash of the content.

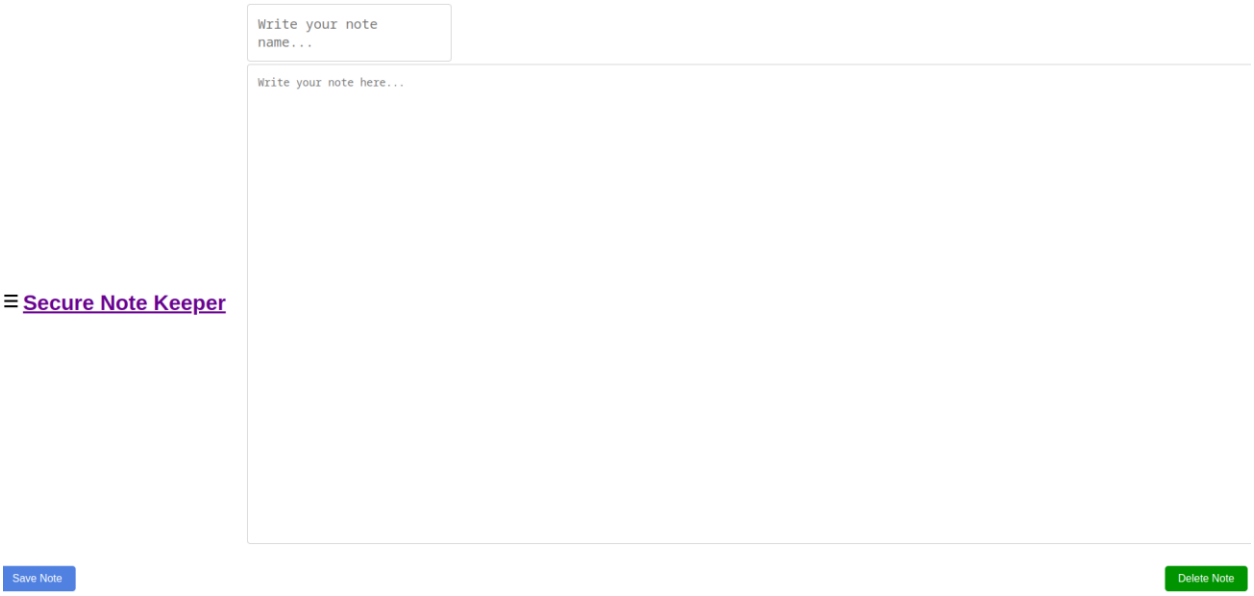
1. Secure note display

2. Secure note display with user profile

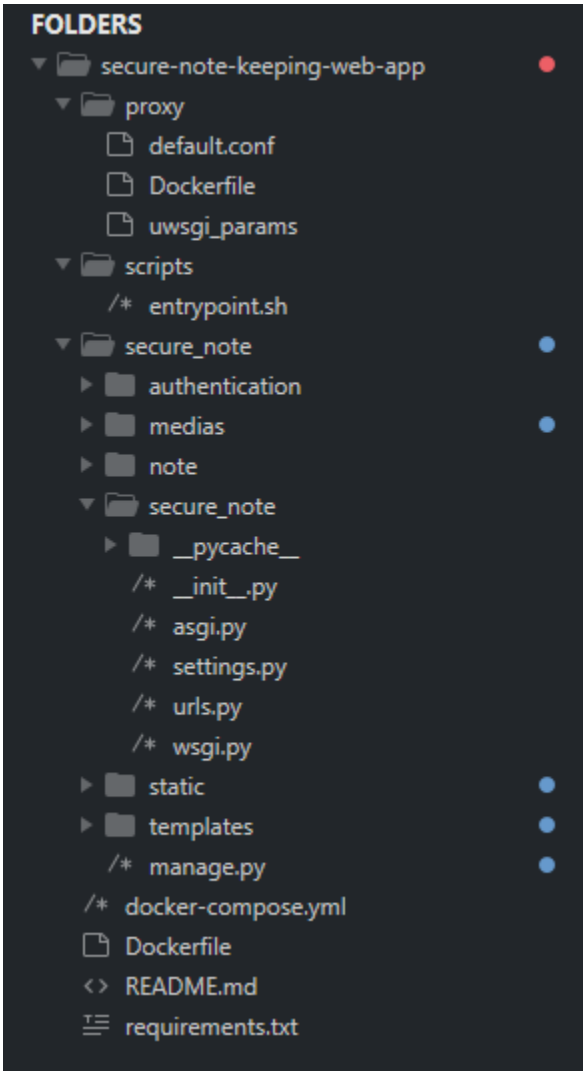
3. *Secure note create with profile*



4. Create note home page



How the code is organized



1. Proxy

1.1. default.conf

default.conf is a nginx file that contain the basic configuration for the nginx server. In this project we make it only have a serve configuration with five things which are

i. **The port the server listens on which are port 80 and 443** but 80 will only use to make backward compatibility and permanently redirect user to https

ii. **Basic configuration of self-signed ssl certificate** : In this project we have encrypted the traffic between nginx server and the client using the self-signed certificate which was generated using the openssl tool

iii. It also define the **maximum data size that the server can handle and accept** if it give beyond 10 MB that it will return a 403 error to the client.

iv. **The static file location** : to server static file it use /static url path and if it get so it will look for that inside the /vol/static

v. **The location of the Django server** to redirect the http request to Django using the uwsgi middleware

```
server{
    listen 80;
    return 301 https://127.0.0.1;
}

server{
    listen 443 ssl;
    ssl_certificate /etc/nginx/ssl/secure_note.crt;
    ssl_certificate_key /etc/nginx/ssl/secure_note.key;

    client_max_body_size 10m;

    location /static {
        alias /vol/static;
    }

    location / {
        uwsgi_pass secure_note:8000;
        include /etc/nginx/uwsgi_params;
    }
}
```

1.2. Dockerfile

As the Docker documentation say it is a line by line code that Docker will execute in a pre-defined image when building.

Based on this we have used this file to make folders for static files, install openssl and make a self-signed certificate using openssl. The image we use from Docker-hub is called **nginxinc/nginx-unprivileged:1-alpine** which is a highly secure image and very light weight. Then we install openssl then we create folder to store the self-signed certificate and static files. We use openssl to generate a self-signed certificate then we delete the app openssl to make our image as lightweight as possible.

```
FROM nginxinc/nginx-unprivileged:1-alpine

COPY ./default.conf /etc/nginx/conf.d/default.conf
COPY ./uwsgi_params /etc/nginx/uwsgi_params

USER root
RUN apk add --update --no-chahe --virtual .tmp openssl
RUN mkdir -p /vol/static
RUN chmod 755 /vol/static -R
RUN mkdir -p /etc/nginx/ssl
RUN chmod 755  /etc/nginx/ssl -R

RUN openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/nginx/ssl/secure_note.key -out /etc/nginx/ssl/
secure_note.crt -subj "/C=ET/ST=Amhara/L=Bahir Dar/O=Bahir dar/CN=mintesnot.afework"

RUN apk del .tmp
# USER nginx
```

1.3. Uwsgi_params

uwsgi_params and uWSGI play a crucial role in deploying Django applications. Let's break down their uses and what they are:

- uWSGI: It is an application server container that implements the Web Server Gateway Interface (WSGI) specification. uWSGI is used to serve Python applications like Django on the web. It acts as a bridge between the web server (like nginx or Apache) and the Python application. It handles client requests, runs the application, and sends responses back to the client
- uwsgi_params: This typically refers to a configuration file used by servers like nginx to pass requests to the uWSGI server when using the uwsgi protocol. It contains parameters that define how the web server communicates with uWSGI, ensuring that the request is properly handled and forwarded to your Django application.

Here's a simplified explanation of how they work together in a Django deployment:

- ✓ A web client makes a request to a web server (like nginx).
- ✓ The web server reads the uwsgi_params to determine how to communicate with the uWSGI server.
- ✓ The web server sends the request to the uWSGI server using the uwsgi protocol.
- ✓ uWSGI processes the request using your Django application, generates a response, and sends it back to the web server.
- ✓ The web server then sends the response back to the web client.

This setup is popular because it combines the strengths of nginx (like handling static files and managing client connections) with the ability to run dynamic Python applications through uWSGI. It's a robust and scalable way to serve Django applications to users.

2. Scripts

2.1. entrypoint.sh

As we can see from the image it is a sh script that will be executed in the python image which we will see later. It basically collect static files like CSS, JavaScript files and media files. Then it make migration to the backend postgres database the models that we have defined in the Django project. Finally the ./note/cryptoengine.py will generate a private and public key using RSA found in Django secure note project which we will see later in more detail. Finally it will start the Django app and listen the uwsgi server to send request at port 8000.

```
#!/bin/sh

set -e

python manage.py collectstatic --no-input
python manage.py makemigrations --no-input
python manage.py migrate --no-input
python ./note/cryptoengine.py --no-input

uwsgi --socket :8000 --master --enable-threads --module secure_note.wsgi
```

3. docker-compose.yml

As we can see from the image we have used three images which are the python, postgres and nginx server image. All of them are alpine Linux based for the seek of security and weight. Every image in the service has the location of the docker file environmental variables and volumes if they have. Some also have dependencies.

The first image is the python image which describes the dockerfile found in the folder where this file exist and where are the static files found to make them sharable between proxy and python image. I also define it depend on postgres image which defined below.

```
services:
  secure_note:
    build:
      context: .
    volumes:
      - static_data:/vol/web
    environment:
      - SECRET_KEY=SAMPLEKEY
    depends_on:
      - pgdb

  pgdb:
        image: postgres:alpine
        container_name: pgdb
        environment:
            - POSTGRES_DB=postgres
            - POSTGRES_USER=postgres
            - POSTGRES_PASSWORD=postgres
        volumes:
            - pgdata:/var/lib/postgresql/data

  proxy:
    build:
      context: ./proxy
    volumes:
      - static_data:/vol/static
    ports:
      - "443:443"
      - "80:80"
    depends_on:
      - secure_note

volumes:
  static_data:
  pgdata:
```

The second one is the postgres image which the image type we are going to use because of no need for additional configuration to the image we do not have any docker file. But it has environmental variables which are defined on the official postgres image documentation.

The third one is the proxy server which has a docker file which is found in the proxy folder as we have seen in the above. We also define or expose ports to outside for the nginx image which are 443 and 80. It also says it is dependent on the python image which we call secure_note.

Finally we define the volumes used and need to be created locally for persistence of data. To view this we can use the docker volume ls command if we build and run the docker engine we will see this two static volume files may be with some name difference.

4. Dockerfile

```
FROM python:3-alpine

ENV PATH="/scripts:${PATH}"

WORKDIR /app
RUN apk add --update --no-cache --virtual .tmp gcc libc-dev linux-headers

COPY  requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
RUN apk del .tmp

COPY ./secure_note/ /app/
WORKDIR /app/

COPY ./scripts /scripts

RUN chmod +x /scripts/*

RUN mkdir -p /vol/web/static/password_rsa
RUN mkdir -p /vol/web/media
COPY ./secure_note/medias /vol/web/media

RUN adduser -D user
RUN chown -R user:user /vol
RUN chown -R user:user /app

RUN chmod -R 755 /vol/web
RUN chmod -R 755 /app


USER user
CMD ["entrypoint.sh"]
```

This docker file is used to build the python image that we will use to install our Django application. If we look at the docker hub repository this python image totally managed by the docker developers and it contain python pre-configured for different use like datascience studies, building python based application like Django and more. This file will basically update an environmental variable for adding the entryscripts file in the scripts folder to PATH. Then it will copy all the necessary files like static files, secure note Django app, media files and scripts to the image. Then we will install all necessary requiments for our Django app like rsa, Django and many more using pip. After That it will set necessary permissions for file access and write operation. Finally it will create a user to make our image run in a less privileged user for security reason and run the entryscript file that we have copied and added to environmental variable at the beginning of the dockerfile.

5. requirments.txt

```
asgiref==3.8.1
Django==5.0.3
pillow==10.3.0
psycopg2-binary==2.9.9
sqlparse==0.4.4
uWSGI==2.0.24
pyasn1==0.6.0
pycryptodome==3.20.0
rsa==4.9
```

This are the list of requirements that our application relays on to work properly and perform different operation like encryption and decryption, generate a private and public key using rsa , work with image, work with Django and forward any response to an http server or web serve like ngnix and apache using uwsgi.

6. Secure note Django application

In this documentation we will try to look at the basic files and folders found in this project like the settings.py, urls.py, forms.py, models.py and views.py. For this modules we will give a detail description but for others we will have a brief explanation. We will cover modules that we design specifically first and proceed to the logic of Django.

1. User-defined modules

In this project we have designed two modules for validation and cryptography. The first one is called validation.py which found in authentication application. The second one is found in the note application and is called crytpoengine.py.

1.1. Validation.py

```python
import re

def validate_password(password:str) -> bool:
  pattern = "^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-]).{8,}$"
  if not re.search(pattern, password):
    return False
  return True
```

```python
def is_all_char(message:str) -> bool:
  return all(i.isalpha() for i in message)

def is_all_char_num(message:str)->bool:
  return all(i.isalpha() or i.isdigit() for i in message)
```

This code will validate the password entered by users when they register and reset password. As we can see this statement will return True or valid if the password contain special character, character and number at least with length of eight.

The second image show the codes used to make firstname , lastname and usename validation and it will return true if a message contain character in the first function and true if a message contains character and number in the second function otherwise false for both

1.2. Cyptoengine

This code is found in the note application and it is used to make different cryptographic operations real like AES, RSA, hashing with md5 and hashing with sha-256. In this part we will try to see this algorithms and how we used them to make notes from users secure and protected.

1.2.1. MessageDigist

```
class MessageDigest:
    @staticmethod
    def md5_hash(message : str) ->  str:
        cipher = hashlib.md5()
        cipher.update(message.encode())
        return cipher.hexdigest()

    @staticmethod
    def sha256_hash(message : str) -> str:
        cipher = hashlib.sha256()
        cipher.update(message.encode())
        return cipher.hexdigest()
```

The message digits is class that contain static methods for performing md5 and sha256 hashing using the hashlib from the builtin modules of python. Both of them work in the same way which is the instate the hashing algorithm as the hashlib developer calls it and add or update the message by adding the byte form of the message then it returns the hex form of the digested message.

1.2.2. RSA

1.2.2.1. Key generation

```
class RSACryptography:
    @staticmethod
    def server_key_generation():
        public,private = rsa.newkeys(2048)
        os.makedirs("/vol/web/static/password_rsa/server/")
        with open("/vol/web/static/password_rsa/server/cryptography_file_for_server.public","wb") as f:
            f.write(public.save_pkcs1("PEM"))
        with open("/vol/web/static/password_rsa/server/cryptography_file_for_server.private","wb") as f:
            f.write(private.save_pkcs1("PEM"))

    @staticmethod
    def key_generation(filename:str) -> bool:
        public,private = rsa.newkeys(1024)
        root_filepath= "/vol/web/static/password_rsa/"
        file_path = root_filepath + filename
        if not os.path.exists(file_path):
            os.makedirs(file_path)
            with open(file_path + "/cryptography_file.public","wb") as f:
                f.write(public.save_pkcs1("PEM"))
            with open(file_path + "/cryptography_file.private","wb") as f:
                f.write(private.save_pkcs1("PEM"))
            return True
        else:
            return False
```

The above algorithm show the RSA implementation of public and private key cryptography that we have used in our project to generate key and perform encryption-decryption and sign-verify. We have two static methods to generate key the one is for the server or python image and the other one is for users. Both of them work in the same way except the file path used by users is given by the web application or generated in some way.

For all we use rsa module to do so and let us look at the work flow of the code:

1. We generate private and public key with 2048 bit length for server and 1024 length for user. We know it is recommended to use 2048 but we have to consider who the user is and we need to threat model. Based

on that it is ok to use 1024 key for users. But the server is the core so we need to protect it in higher priority and cost.

2. Then we will create the directory to store the key in as a file
3. Then we will write our public and private key to files in PEM(privacy enhanced mail) format
4. For the user side we check, if the folder exist and return false if it do so to avoid any overwrite of key

1.2.2.2. Encryption and Decryption

```
@staticmethod
def encryption(filename : str,message:bytes) -> bytes:
    root_filepath= "/vol/web/static/password_rsa/"
    file_path = root_filepath + filename
    with open(file_path + "/cryptography_file.public","rb") as f:
        public_key = rsa.PublicKey.load_pkcs1(f.read())
    result = rsa.encrypt(message,public_key)
    return result

@staticmethod
def decryption(filename : str,cipher:bytes) -> bytes:
    root_filepath= "/vol/web/static/password_rsa/"
    file_path = root_filepath + filename
    with open(file_path + "/cryptography_file.private","rb") as f:
        private_key = rsa.PrivateKey.load_pkcs1(f.read())
    result = rsa.decrypt(cipher,private_key)
    return result
```

This is the cryptographic implementation of RSA to encrypt and decrypt using the user public and private key. As we can see first it will always accept a filename to identify the private and public folder for that specific user then it fetch the key based on the operation means for encryption we use public key and for decryption we use private key. This will protect the confidentiality of the data being encrypted and decrypted. But to ensure that the data is not modified and it is stored by serve we use the concept of signing which we will see below.

let us look at the work flow of the code:

- ✓ We accept the filename or the folder name after /vol/web/static/password_rsa
- ✓ We read the private or public key based on the path
- ✓ Then we perform encryption by giving byte for of the message and the public key
- ✓ Then we return the byte form
- ✓ We do the same for decryption except we use only the private key to do decryption

1.2.2.3. Signing and verifying

We use this part to ensure the integrity and non-repudiation of the data being signed. In our scenario we use this to sign the encrypted key using the public key of the user and we sign the sha1 of the encrypted key. When want to fetch the key from the database we use the verify method to check the validity of the key. But the above will sign the message using the private key of the server and we verify it using the public key which return hashing algorithm if it is valid if not it will raise rsa.verification error. Based on that we use try and except to return true if valid and false otherwise.

```python
@staticmethod
def sign(message : bytes) -> bytes:
    with open("/vol/web/static/password_rsa/server/cryptography_file_for_server.private","rb") as f:
        private_key = rsa.PrivateKey.load_pkcs1(f.read())
    result = rsa.sign(message,private_key,"SHA-1")
    return result

@staticmethod
def verify_sign(message: bytes,signature:bytes) -> bool:
    with open("/vol/web/static/password_rsa/server/cryptography_file_for_server.public","rb") as f:
        public_key = rsa.PublicKey.load_pkcs1(f.read())
    try:
        rsa.verify(message,signature,public_key)
        return True
    except rsa.VerificationError:
        return False
```

Let us look at the set by step guide:

- ✓ As it was in encryption-decryption we first fetch the private or public key of the server to do the operation in hand
- ✓ We sign the byte message using the rsa.sign method and to do so we give the message, private key and the hashing algorithm and it will return the signature of the message
- ✓ To verify we use the verify method by passing message, signature and public key then it returns the hashing algorithm if not it raise rsa.verficationerror

1.2.3. AES

```python
class AESCryptography:
    @staticmethod
    def key_generation(password : str) -> bytes:
        salt = get_random_bytes(32)
        secret_key = PBKDF2(password,salt,dkLen=32)
        return secret_key

    @staticmethod
    def encryption(secret_key:bytes,message : bytes) -> bytes:
        cipher = AES.new(secret_key,AES.MODE_CBC)
        result = cipher.iv + cipher.encrypt(pad(message, AES.block_size))
        return result

    @staticmethod
    def decryption(message : bytes,secret_key:bytes) -> bytes:
        iv = message[:16]
        cipher_text = message[16:]
        cipher = AES.new(secret_key,AES.MODE_CBC,iv=iv)
        orginal = unpad(cipher.decrypt(cipher_text), AES.block_size)
        return orginal
```

1.2.3.1. Key generation

The first step is to generate a symmetric key to do the cryptographic operation. To do so we have used the pycrpto module which is now called pycryptodome and from that we imported modules and classes that are useful to do different tasks.

Let us look at the step by step guide to the process

- ✓ We generate a random 16 byte data to use it as a salt which is useful for making our system resistant to dictionary and rainbow table attacks

- ✓ Then we use pbkdf2(password based key generation function version 2) to generate the key by giving string password, salt and the size expected we can also give a count which specify the iteration to make it more scrammbled
- ✓ Finally we return back the key

1.2.3.2. Encryption

In the encryption process we have used the AES module to encrypt and let us look at it step by step:

- ✓ We first instate the AES class by passing the secret key and mode of operation
- ✓ We encrypt the block but to avoid error we pad it based on the AES block size and append it to the randomly generated 16 byte data
- ✓ We return the cipher text

1.2.3.3. Decryption

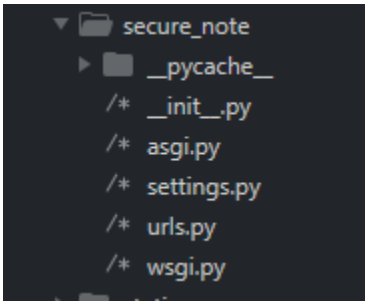
In the decryption process we have used the AES module to encrypt and let us look at it step by step:

- ✓ We take the iv or randomly generated 16 byte of data based on the logic we follow at the encryption process
- ✓ We instate the AES by passing secret key and the mode of operation with the iv
- ✓ We then decrypt the text and apply unpad based on the AES block size in use
- ✓ Then we return the plain text in byte form or we can decode it

2. Django Application Flow

Our Django application contain three apps and tries to fulfill basic two functions which are note management and user authentication. The two functionalities are fulfilled by note and authentication apps respectively. The last one or the

2.1. Secure note application



secure_note application is the one that is used by Django as a main app which contain the settings file, root url path and wsgi file.

The wsgi file is the configuration used by Django to accept connection from uwsgi which is used to make a bridge between python based application and web server like nginx and apache

The root url is the first url file that the project looks first when it get any request to a path and it defined in the settings file but by default it is the urls.py file in the main or first application that contain the settings file.

```python
from django.contrib import admin
from django.urls import path,include
from django.conf import settings
from django.conf.urls.static import static

app_name = "main"
urlpatterns = [
    # path('admin/', admin.site.urls,name="admin"),
    path('',include('authentication.urls'),name="index"),
    path("note/",include("note.urls"),name="note"),
]

urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT) #static file addition from the setting file indjango
urlpatterns += static(settings.MEDIA_URL,document_root=settings.MEDIA_ROOT) #media file addition from the setting file indjango
```

2.1.1. settings

This is the application that contain the basic configuration starting from root url resolver, installed apps, security features and other things which are very useful for any web app. From this we will look at the basic security features like the csrf ,session and other related security.

```python
ALLOWED_HOSTS = ["*"]
CSRF_TRUSTED_ORIGINS = ["https://localhost","https://127.0.0.1","https://mintesnot.afework"]
```

This show that it is allowed to any host to access it from any where based on the allowed host configuration

The csrf trusted show that it is allowed only for this domain to send a csrf token it accept from this domains only to avoid cross domain attacks.

```python
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

This configuration contain the middle wares used for security like csrf which we use to generate token and check it at form validation time

```python
######################
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
LANGUAGE_COOKIE_SECURE = True
CSRF_COOKIE_HTTPONLY = True
LANGUAGE_COOKIE_HTTPONLY = True.
```

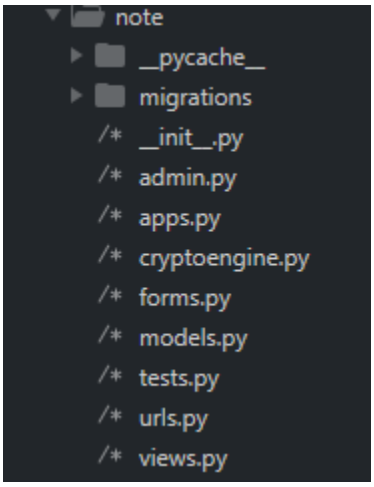
The CSRF_COOKIE_SECURE setting controls whether to use a secure cookie for the CSRF token.
SESSION_COOKIE_SECURE: Set it to True to make session cookies secure.

LANGUAGE_COOKIE_SECURE: Also set it to True to secure language cookies.

CSRF_COOKIE_HTTPONLY: Set to True to prevent client-side JavaScript access to the CSRF cookie.

LANGUAGE_COOKIE_HTTPONLY: Similarly, set it to True for language cookies3.

2.2. Note application



This application contain the core things that we are required to perform which are creating a note, deleting and editing note. As we can it also contain the crypto engine module which we have discussed in the previous section on user defined part. Let us break it all and look at what they do briefly.

Forms is the module that we use to validate user input and check for some properties like size and required or not in the application when a form is submitted.

```
class SaveForm(forms.Form):
    filecontent = forms.CharField(max_length=10000,min_length=1,required=True)
    filename = forms.CharField(max_length=64,min_length=2,required=True)
```

Models contain the file model which we use to define the format we use and the table schema we need to store files.

```
class FileModel(models.Model):
    user = models.ForeignKey(User,on_delete=models.CASCADE)
    name = models.CharField(max_length=64)
    content = models.BinaryField()
    date_time = models.DateTimeField(default=now)
    sha256_hash = models.CharField(max_length=512)
    md5_hash = models.CharField(max_length=128)
```

Urls module contain the path used by the application to make decision on routes which begins with note. Example https://domain/note/{anything}.

This contain the view that we have used to interact with the user and handle requests and give responses.it contain four view classes which are FileDisplay, IndexView, DeleteView and SaveFile.

2.2.1. FileDisplay

This will fetch the database for the files owned by the user then it will look for the specific file name

Then it will fetch the key for aes encryption and decryption then verify the key using the servers public key and decrypt it using the user private key.

Finally it will decrypt the file content using the aes key and send it to the user.

```
class FileDisplayView(LoginRequiredMixin,View):
    login_url = "/login/"
    def get(self,requests,name):
        file = FileModel.objects.filter(user=requests.user).get(name=name)
        list_of_files = FileModel.objects.filter(user=requests.user)
        user_password = UserProfile.objects.get(user_user=requests.user)
        aes_password = cryptoengine.RSACryptography.decryption(user_password.file_path,user_password.hashed_password)
        check_validaty = cryptoengine.RSACryptography.verify_sign(aes_password,user_password.signed_password)
        if check_validaty:
            content = cryptoengine.AESCryptography.decryption(file.content,aes_password)
        else:
            pass
        return render(requests,"note/index2.html",{"file":file,"list_of_file":list_of_files,"content":content.decode(),"user_password":u
```

2.2.2. IndexView

The view page that the user will redirected to when he enter the https:// domain name /note

It will fetch the user profile and the list of files owned by the user and return it to the user using an templates.

```
class IndexView(LoginRequiredMixin,View):
    login_url="/login/"
    def get(self,requests):
        list_of_file = FileModel.objects.filter(user=requests.user)
        user_profile = UserProfile.objects.get(user_user = requests.user)
        return render(requests,"note/index.html",{"list_of_file":list_of_file,"user_password":user_profile})
```

2.2.3. DeleteView

```
class DeleteFileView(LoginRequiredMixin,View):
    login_url = "/login/"
    def get(self,requests,name):
        file = FileModel.objects.filter(user = requests.user).get(name=name)
        return render(requests,"note/delete_confirm.html",{"file":file})

    def post(self,requests,name):
        file = FileModel.objects.filter(user = requests.user).get(name=name)
        file.delete()
        return HttpResponsePermanentRedirect(reverse("note:index"))
```

This will accept get and post requests the get request send when the user click delete button at the home page and we send him a confirmation page if delete it will send post request with the file name

Otherwise it will redirect the user to the home page.

2.2.4. SaveFile

```python
class SaveFile(LoginRequiredMixin,View):
    login_url="/login/"
    def get(self,requests):
        return HttpResponsePermanentRedirect(reverse("note:index"))

    def post(self,requests):
        save_form = SaveForm(data=requests.POST)
        if save_form.is_valid():
            file_name = requests.POST.get('filename')
            file_content = requests.POST.get('filecontent')
            try:
                temp = FileModel.objects.filter(user=requests.user).get(name=file_name)
                raise FileExistsError()
            except FileExistsError:
                m = temp
                user_password = UserProfile.objects.get(user_user=requests.user)
                aes_password = cryptoengine.RSACryptography.decryption(user_password.file_path,user_password.hashed_password)
                check_validaty = cryptoengine.RSACryptography.verify_sign(aes_password,user_password.signed_password)
                if check_validaty:
                    m.content = cryptoengine.AESCryptography.encryption(aes_password,file_content.encode())
                else:
                    pass
                m.sha256_hash = cryptoengine.MessageDigest.sha256_hash(file_content)
                m.md5_hash = cryptoengine.MessageDigest.md5_hash(file_content)
                m.save()
            except:
                m = FileModel()
                m.name = file_name
                user_password = UserProfile.objects.get(user_user=requests.user)
                aes_password = cryptoengine.RSACryptography.decryption(user_password.file_path,user_password.hashed_password)
                check_validaty = cryptoengine.RSACryptography.verify_sign(aes_password,user_password.signed_password)
                if check_validaty:
                    m.content = cryptoengine.AESCryptography.encryption(aes_password,file_content.encode())
                else:
                    pass
                m.user = requests.user
                m.sha256_hash = cryptoengine.MessageDigest.sha256_hash(file_content)
                m.md5_hash = cryptoengine.MessageDigest.md5_hash(file_content)
                m.save()
            return HttpResponsePermanentRedirect(reverse("note:file_display",args = (file_name,)))
        else:
            return Http404()
```

The SaveFile class you've shared is a Django view for handling file saving functionality. It inherits from LoginRequiredMixin and View. The LoginRequiredMixin ensures that only authenticated users can access this view.

- login_url="/login/": This is the URL where users will be redirected if they are not logged in.
- get(self, requests): This method handles GET requests. It redirects the user to the index page of the note app.
- post(self, requests): This method handles POST requests, which are typically form submissions. It validates the submitted save form. If the form is valid, it retrieves the file name and content from the form data. It then tries to retrieve a FileModel object for the current user with the given file name. If such a FileModel object exists, it raises a FileExistsError. In the FileExistsError exception handler, it retrieves the user's password and decrypts it, verifies the decrypted password, encrypts the file content with the decrypted password if the password is verified, calculates the SHA256 and MD5 hashes of the file content, and saves the FileModel object. If no such FileModel object exists, it creates a new FileModel object with the given file name, the current user, the encrypted file content, and the SHA256 and MD5 hashes of the file content, and saves it. It then redirects to the file display page for the given file name. If the form is not valid, it returns a 404 error

2,3. Authentication application

It has the same thing with the note application except the view we will focus on that only.

2.3.1. Forms.py

```
class LoginForm(forms.Form):
    username = forms.CharField(max_length=64,min_length=5,required=True)
    password = forms.CharField(max_length=50,min_length=8,required=True)

class RegistrationForm(forms.Form):
    username = forms.CharField(max_length=64,min_length=5,required=True)
    password1 = forms.CharField(max_length=50,min_length=8,required=True)
    password2 = forms.CharField(max_length=50,min_length=8,required=True)
    email = forms.EmailField(max_length=320,min_length=5)
    firstname = forms.CharField(max_length=64,min_length=5,required=True)
    lastname = forms.CharField(max_length=64,min_length=5,required=True)

class UpdateForm(forms.Form):
    email = forms.EmailField(max_length=320,min_length=5)
    firstname = forms.CharField(max_length=64,min_length=5,required=True)
    lastname = forms.CharField(max_length=64,min_length=5,required=True)
    profile_picture = forms.ImageField(required=True)


class PasswordResetForm(forms.Form):
    password1 = forms.CharField(max_length=50,min_length=8,required=True)
    password2 = forms.CharField(max_length=50,min_length=8,required=True)
```

2.3.2. Views

The views is the one that handle the request and responses and the place where our logic of the application will be found. In this part of the project we have tried to perform the basic logic to register user and log them in and out. It contain eight classes which we will see step by step:

2.3.2.1. Indexview

```
class Index(LoginRequiredMixin,View):
    login_url = "/login"
    def get(self, requests):
        return HttpResponsePermanentRedirect(reverse("note:index"))
```

This is the class that will respond for https://domain_name / requests which will make sure the use is authentic and redirect the request based on that if it is authentic it will redirect to the home page if not to the /login url path

2.3.2.2. LoginView

```
class LoginView(View):
    def get(self,requests):
        return render(requests,"authentication/login/index.html")

    def post(self,requests):
        login_form = LoginForm(data=requests.POST)
        if login_form.is_valid():
            username = requests.POST["username"]
            password = requests.POST["password"]
            user = authenticate(requests, username=username, password=password)
            if user is not None:
                login(requests,user)
                return HttpResponsePermanentRedirect(reverse("note:index",))
        message = "Invalid username or password"
        return render(requests,"authentication/login/index.html",{"message" : message})
```

This will only accept get and post requests and based on that it will respond. For the get request it will give the login page and for the post it will accept the form and validate the rightness of the form using the pre-defined form class in forms.py module if valid it will fetch the username and password then authenticate using he Django pre-defined authenticate method then is the username and password are right it will return an object otherwise None. If we get an object we will proceed to logging in the user and redirecting him to the home page. Otherwise it will return the user the login page with an error message.

2.3.2.3. RegisterationView

```
class RegistrationView(View):
    def get(self,requests):
        return render(requests,"authentication/login/index.html")

    def post(self,requests):
        registeration_form = RegistrationForm(data=requests.POST)
        if registeration_form.is_valid():
            message=""
            if not (validate_password(requests.POST.get("password1"))):
                message = "The password is invalid\nMake sure it has captial and small letter with number and special character"
            elif not requests.POST.get("password1") == requests.POST.get("password2"):
                message = "The password does not match"
            elif not is_all_char(requests.POST["firstname"] + requests.POST["lastname"]):
                message = "The name is invalid only use character"
            elif not is_all_char_num(requests.POST["username"]):
                message = "The username is invlaid only use alphabets and digit"
            else:
                try:
                    user_temp = User.objects.get(username=requests.POST.get("username"))
                except:
                    user = User.objects.create_user(requests.POST.get("username"), requests.POST.get("email"), requests.POST.get("pass
                    user.first_name = requests.POST.get("firstname")
                    user.last_name = requests.POST.get("lastname")
                    user.save()
                    file_path = cryptoengine.MessageDigest.sha256_hash(user.username + str(random.randbytes))
                    while len(UserProfile.objects.filter(file_path = file_path)) != 0:
                        file_path = cryptoengine.MessageDigest.sha256_hash(user.username + str(random.randbytes))
                    cryptoengine.RSACryptography.key_generation(file_path)
                    aes_secret_key = cryptoengine.AESCryptography.key_generation(requests.POST.get("password1"))
                    singed_aes_key = cryptoengine.RSACryptography.sign(aes_secret_key)
                    aes_rsa_encrytion = cryptoengine.RSACryptography.encryption(file_path,aes_secret_key)
                    user_profile = UserProfile(user_user=user,signed_password=singed_aes_key, hashed_password=aes_rsa_encrytion,file_p
                    user_profile.save()
                else:
```

This also has two methods which are get and post. As the above one if it get a get request it will send the registeration page and accept the post request with the data it has client side validation but to make sure it is secure we have used a validation method from valdation module to make sure the password fulfill the requiements. Then we check the similarity of the passwords then validate username, firstname and lastname after that tried to fetch user with a given username if it exist it will execute correctly and pass to the else part if not it will generate an error which will handled by us to create a user with his profile. Also we will generate the key for

symmetric and asymmetric encryption and decryption for the user only. Then store the public and private in pem file at the python image or server and save the encrypted and signed form of the aes key in the postgres database.

2.3.2.4. LogoutView

```
lass LogoutView(LoginRequiredMixin,View):
   login_url="/login/"
   def get(self,requests):
        logout(requests)
        return HttpResponsePermanentRedirect(reverse("authentication:index",))
```

This view is used to check the validity of the user or is the user authentic then will log him out based on that. Then redirect him to the login page.

2.3.2.5. UpdateView

```
class UpdateView(LoginRequiredMixin,View):
    login_url = "/login/"
    def get(self,requests):
        user = User.objects.get(id = requests.user.id)
        user_profile = UserProfile.objects.get(user_user = user)
        return render(requests,"authentication/reset/index.html",{"user":user,"user_profile":user_profile})

    def post(self,requests):
        user = User.objects.get(id = requests.user.id)
        user_profile = UserProfile.objects.get(user_user = user)
        update_form = UpdateForm(data=requests.POST)
        if not is_all_char(requests.POST["firstname"] + requests.POST["lastname"]):
            return Http404()
        elif update_form.is_valid():
            user.first_name = requests.POST.get("firstname")
            user.last_name = requests.POST.get("lastname")
            user.email = requests.POST.get("email")
            user_profile.profile_picture = requests.FILES['profile_picture']
            user.save()
            user_profile.save()
            return HttpResponsePermanentRedirect(reverse("note:index",))
        return render(requests,"authentication/reset/index.html",{"user":user,"user_profile":user_profile,"form":update_form.errors})
```

It in*herits from LoginRequiredMixin and View. The* Log*inRequiredMixin* ensures that only authenticated users can access this view.

- *login_url = "/login/"*: This is the URL where users will be redirected if they are not logged in.

- *get(self, requests)*: This method handles GET requests. It retrieves the current user and their profile, then renders the profile update page with the user and user profile data.

- *post(self, requests)*: This method handles POST requests, which are typically form submissions. It retrieves the current user and their profile, then validates the submitted update form. If the form is valid, it checks whether the first name and last name fields contain only characters. If they do, it updates the user's first name, last name, email, and profile picture with the submitted data, saves the user and user profile, then redirects to the index page of the note app. If the form is not valid or the first name and last name fields contain non-character data, it renders the profile update page with the user, user profile, and form errors.

2.3.2.6. PasswordReset

```
class PasswordRestView(LoginRequiredMixin,View):
    login_url = "/login/"
    def get(self,requests):
        user = User.objects.get(id = requests.user.id)
        user_profile = UserProfile.objects.get(user_user = user)
        return render(requests,"authentication/reset/index.html",{"user":user,"user_profile":user_profile})

    def post(self,requests):
        password_reset_form = PasswordResetForm(data=requests.POST)
        if password_reset_form.is_valid():
            message = ""
            if not (validate_password(requests.POST.get("password1"))):
                message = "The password is invalid\nMake sure it has captial and small letter with number and special character"
            elif requests.POST.get("password1") == requests.POST.get("password2"):
                user = requests.user
                user.set_password(requests.POST.get("password1"))
                user.save()
                return HttpResponsePermanentRedirect(reverse("note:index",))
            elif requests.POST.get("password1") != requests.POST.get("password2"):
                message = "The password field is not the same"
            elif requests.POST.get("password1") == "":
                message = "The password field is required"
        else:
            message = password_reset_form.errors
        user = User.objects.get(id = requests.user.id)
        user_profile = UserProfile.objects.get(user_user = user)
        return render(requests,"authentication/reset/index.html",{"message":message,"user":user,"user_profile":user_profile})
```

This class is a Django view for handling password reset functionality. It inherits from LoginRequiredMixin and View. The LoginRequiredMixin ensures that only authenticated users can access this view.

- login_url = "/login/": This is the URL where users will be redirected if they are not logged in.

- get(self, requests): This method handles GET requests. It retrieves the current user and their profile, then renders the password reset page with the user and user profile data.

- post(self, requests): This method handles POST requests, which are typically form submissions. It validates the submitted password reset form. If the form is valid, it checks the new password against certain conditions (e.g., whether it contains a mix of uppercase and lowercase letters, numbers, and special characters, whether the two password fields match, etc.). Depending on these checks, it sets an appropriate message. If the new password is valid and the two password fields match, it sets the new password for the user and saves it, then redirects to the index page of the note app. If the form is not valid, it sets the form errors as the message. Finally, it renders the password reset page with the message, user, and user profile data.

This view provides a secure way for users to reset their passwords, with checks for password strength and matching password fields. It also ensures that only the authenticated user can reset their password.

2.3.2.7. Forget and About views

```
class ForgetView(View):
    def get(self,requests):
        return render(requests,"authentication/forget/index.html")

    def post(self,requests):
        return HttpResponsePermanentRedirect(reverse("authentication:index",))


class AboutView(View):
    def get(self,requests):
        return render(requests,"bio/index.html")
```

The Forget view is not implemented due to lack of internet access but the about page is the one that return the creator and developers of this project.

3. Templates

Django templates are text documents or Python strings that are marked up using the Django Template Language (DTL). They contain both static parts of the desired HTML output and special syntax for how dynamic content will be inserted.

Django's template engine provides a powerful mini-language for defining the user-facing layer of your application, encouraging a clean separation of application and presentation logic. This means that templates can be maintained by anyone with an understanding of HTML; no knowledge of Python is required

In this project we have mainly around seven templates but for this documentation will focus on templates with forms. In our template all forms has a **csrf_token** which is used to protect our form from cross site request forgery and we apply a **escapejs** which means every element will be checked for javascript content and we make sure it is not added directly which will protect our web site from xss attacks also we used **patterns** to make sure the user enters the required things no more no less.

We can look at this in all but we have taken the form in templates/reset/index.html

```
<form action="{% url 'authentication:update' %}" method='post' class="sign-up-form" enctype="multipart/form-data">
  {% csrf_token %}
  <h2 class="title">Update Profile</h2>
  <div class="input-field">
    <i class="fas fa-user"></i>
    <input type="file" placeholder="Profile Picture" name="profile_picture" lable="profile picture" value=
    {{user.profile_picture|escapejs}} accept="image/*" required />
  </div>
  <div class="input-field">
    <i class="fas fa-user"></i>
    <input type="text" placeholder="Firstname" name="firstname" lable="First name" minlength="5"  maxlength="64" value=
    {{user.first_name|escapejs}} pattern="[a-zA-Z]+" required />
  </div>
  <div class="input-field">
    <i class="fas fa-user"></i>
    <input type="text" placeholder="Lastname" name="lastname" lable="Last name" minlength="5" maxlength="64" value=
    {{user.last_name|escapejs}} pattern="[a-zA-Z]+" required/>
  </div>
  <div class="input-field">
    <i class="fas fa-envelope"></i>
    <input type="email" placeholder="Email" name="email" lable="Email" minlength="5" maxlength="320" value=
    {{user.email|escapejs}} required/>
  </div>
  <input type="submit" value="update profile" class="btn solid" />
</form>
```

4. Media and static files

This are the one that contain the static files like javascript and css with the image we have used in as media.

Features

User Authentication : it contain a secure login, logout, registration, update, reset and forget page

Note Management : the user is able to create, delete and update his note if authentic

Data Encryption : it will encrypt the data by the user AES key and store it in binary form in the database

Key Management : the application encrypt the users AES key using the user public key and sign it with the server private key and store it in the database. But the private and public key combination will never be sent to the database it will be stored in pem format at the Django application server which is called python Docker image.

Secure Communication : we have implemented a self-signed certificate at the nginx server side to reduce the load on the python image that contain the Django application and also the Django application is configured to respond request from an ssl based traffic.

Reference

1. https://docs.djangoproject.com/en/5.0/
2. https://www.w3schools.com/
3. https://docs.djangoproject.com/en/5.0/topics/security/
4. https://stuvel.eu/python-rsa-doc/index.html
5. https://pycryptodome.readthedocs.io/en/latest/
6. https://gemini.google.com/app
7. https://hub.docker.com/_/postgres
8. https://hub.docker.com/_/python
9. https://hub.docker.com/_/nginx