Table of Contents

[**Introduction** 2](#_Toc165150272)

[**Installation guide** 4](#_Toc165150273)

[1. Requirements 4](#_Toc165150274)

[2. Installation 4](#_Toc165150275)

[**How to interact with** 7](#_Toc165150276)

[**1.** **Login page** 7](#_Toc165150277)

[**2.** **Registration Page** 7](#_Toc165150278)

[**3.** **Forget Page** 8](#_Toc165150279)

[**4.** **Update Page** 8](#_Toc165150280)

[**5.** **Reset page:** 9](#_Toc165150281)

[**6.** **About Us Page** 9](#_Toc165150282)

[**7.** **Home Page** 10](#_Toc165150283)

[**How the code is organized** 12](#_Toc165150284)

[**1.** **Proxy** 12](#_Toc165150285)

[1.1. default.conf 12](#_Toc165150286)

[1.2. Dockerfile 13](#_Toc165150287)

[1.3. Uwsgi\_params 14](#_Toc165150288)

[**2.** **Scripts** 14](#_Toc165150289)

[2.1. entrypoint.sh 14](#_Toc165150290)

[**3.** **docker-compose.yml** 15](#_Toc165150291)

[**4.** **Dockerfile** 17](#_Toc165150292)

[**5.** **requirments.txt** 18](#_Toc165150293)

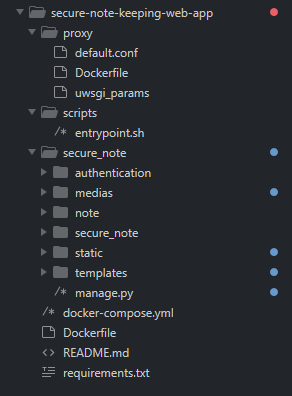
[**6.** **Secure note Django application** 18](#_Toc165150294)

[1. User-defined modules 18](#_Toc165150295)

[2. Django Application Flow 23](#_Toc165150296)

# **Introduction**

In this assignment we tried to make a secure 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 last two 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**

## Requirements

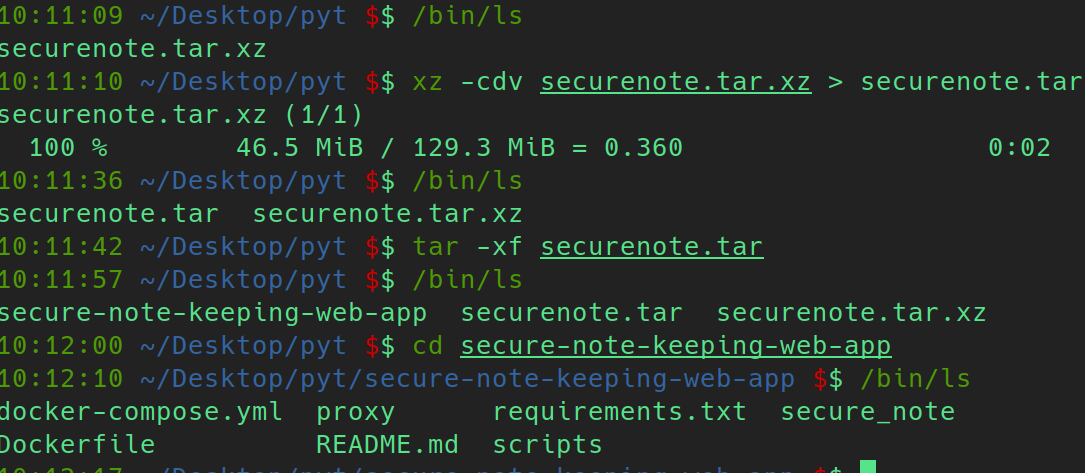
Before the installation there are things that are needed to be fulfilled before doing anything with this web application.

* 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 **()** and the link for Docker configuration guide from Docker developers **().**
* 200 MG free space

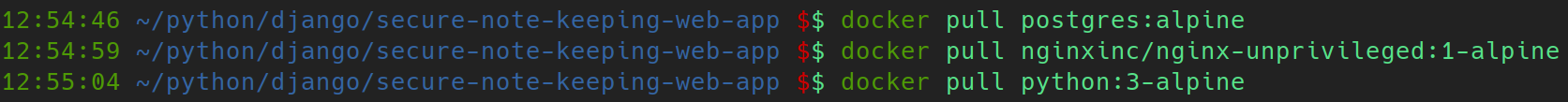
## Installation

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.



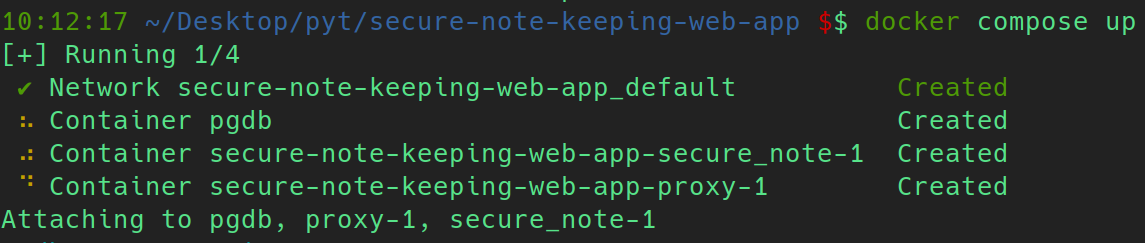
1. To optimize our time effectively it is recommended to do this step but it is not mandatory we can skip it. This step will make a pull from the docker hub to get every required image for this assignment

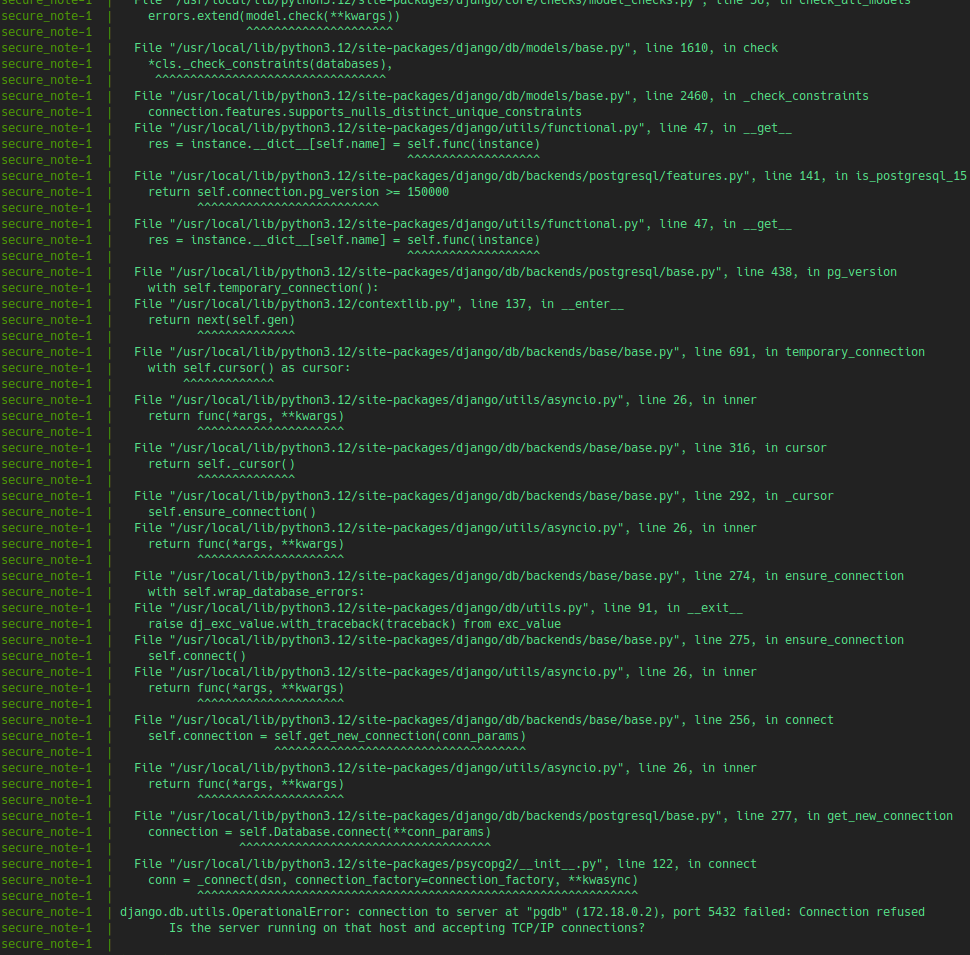


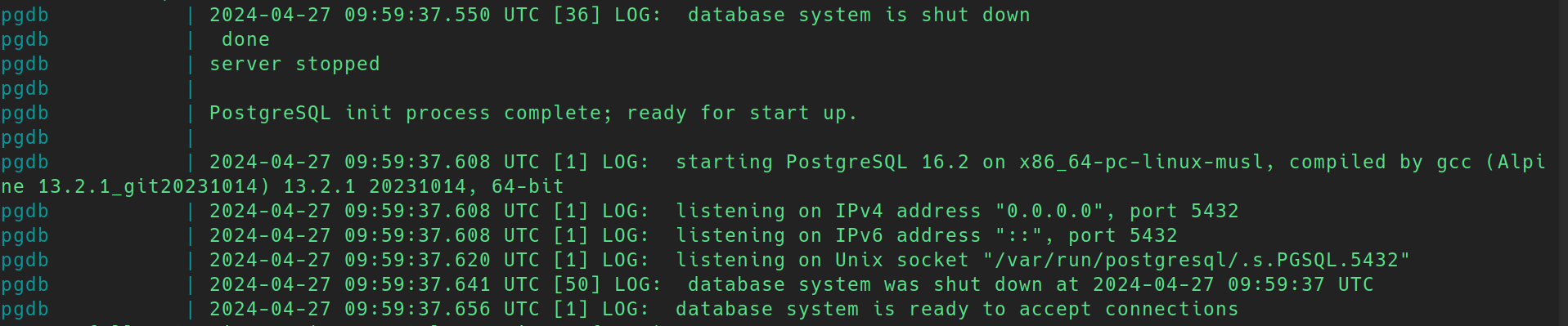
1. Then run the above command to extract



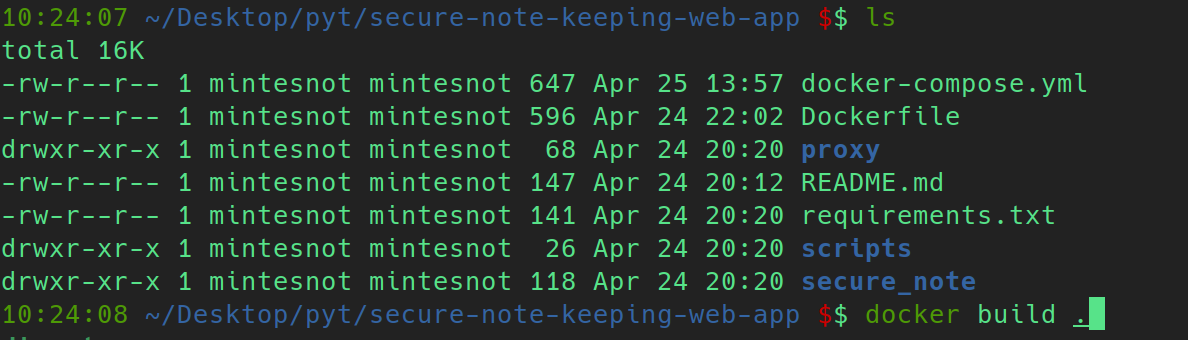
1. Then change directory to the extracted folder and run the docker compose up --build as show in the above.



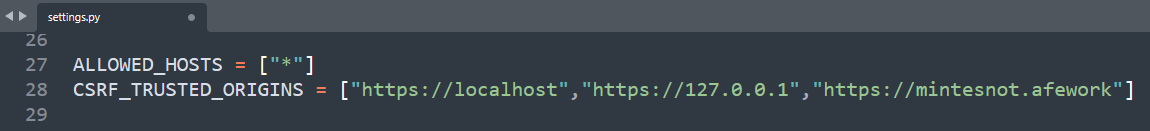




1. If it generate error like that it is because Django tries to migrate before instating the Postgres database so wait for few second until you see the above message Ctrl + C to exit and run docker compose up.



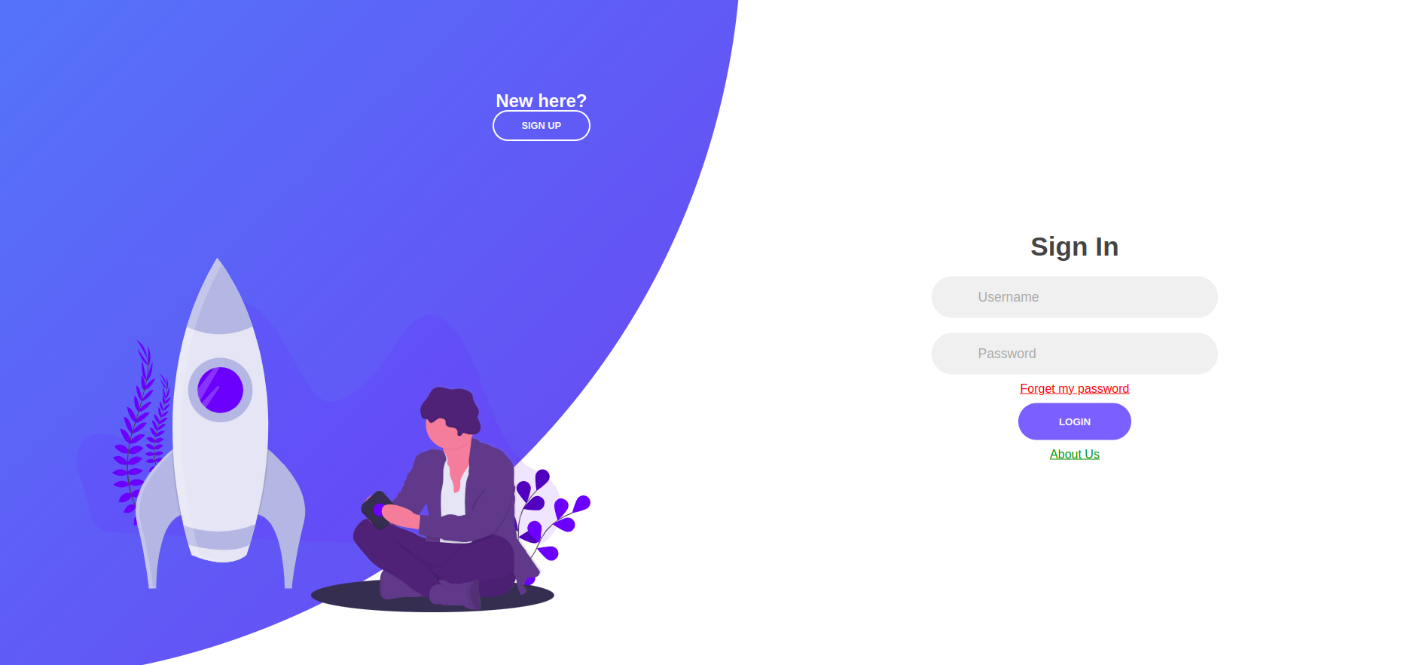
1. Build if like to build the image first and run the app use the above option run using the set 4 option.



1. 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)
2. 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).
3. Conguradulations you have finished the first phase.

# **How to interact with**

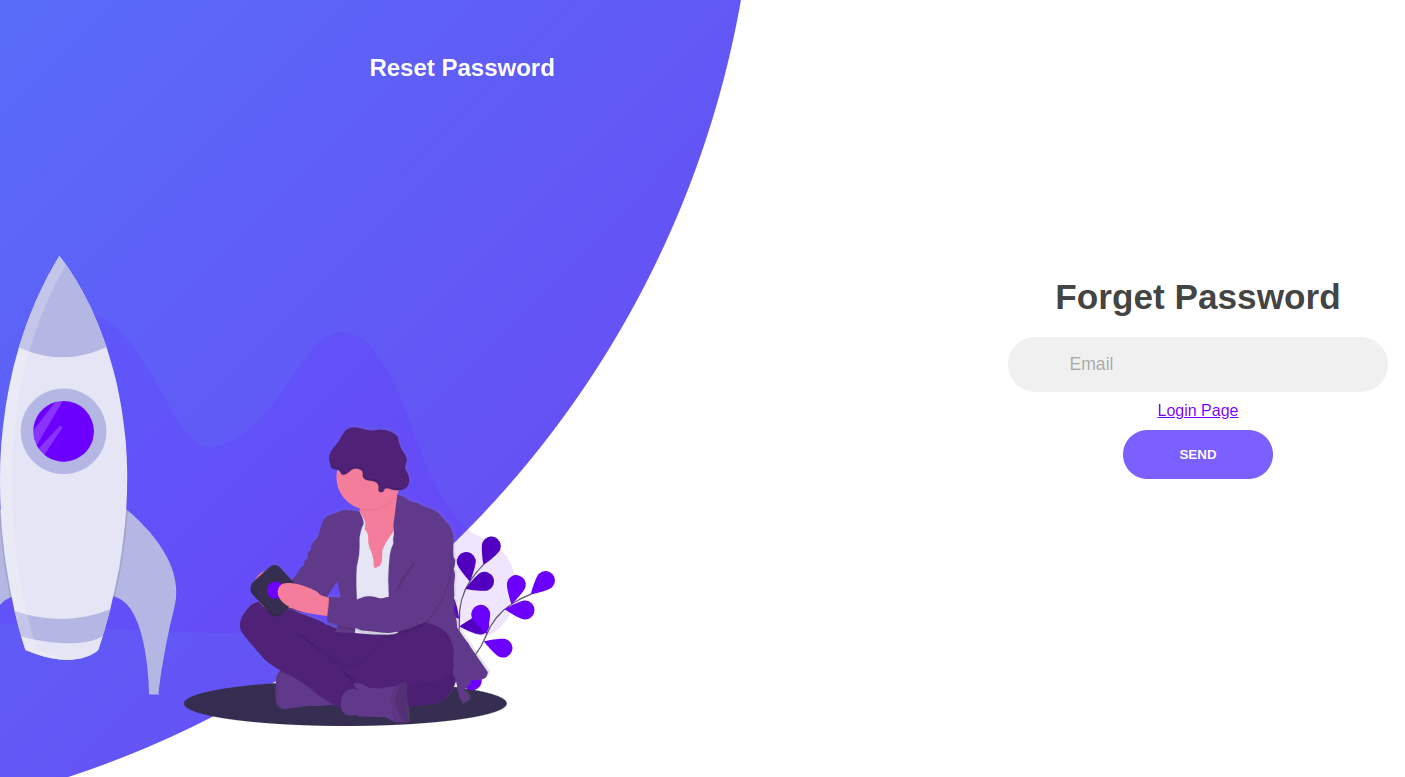
1. **Login page**: is the page used to accept the registered username and password to create them a live session by authenticating them.



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



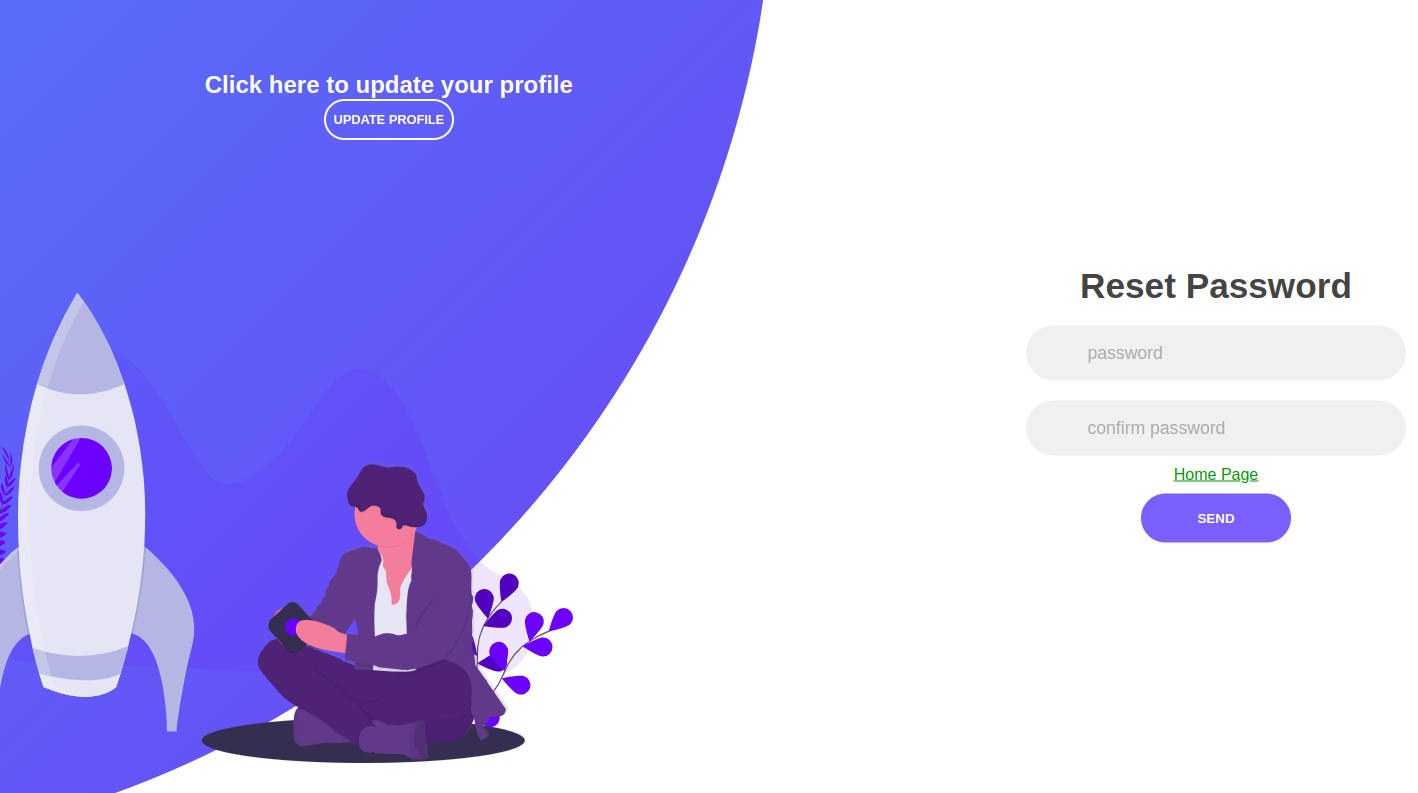
1. **Forget Page**: is the page used to reset password using email



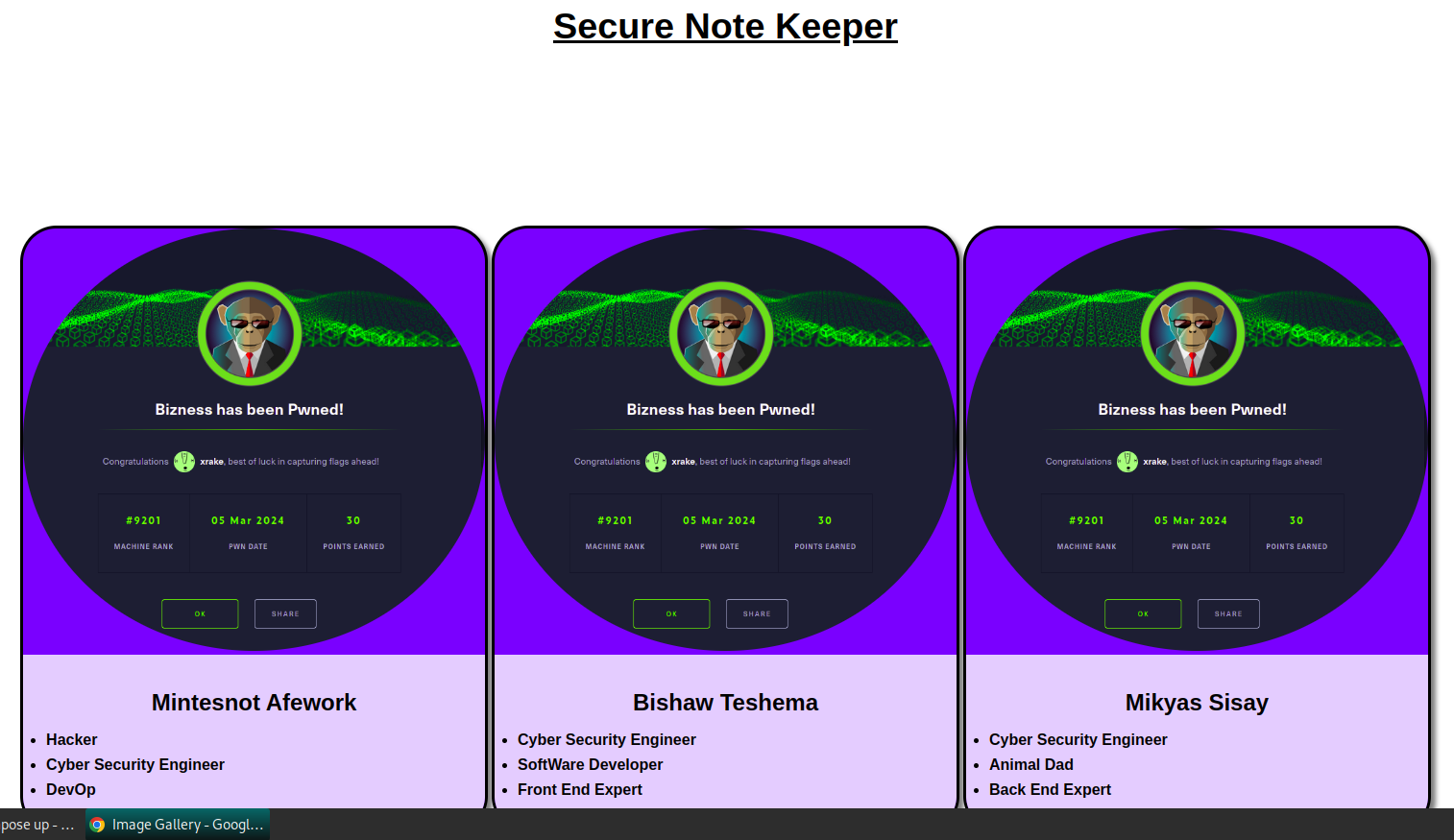
1. **Update Page**: is used to update user profile picture, first name, last name and email



1. **Reset page:** is used to change our password



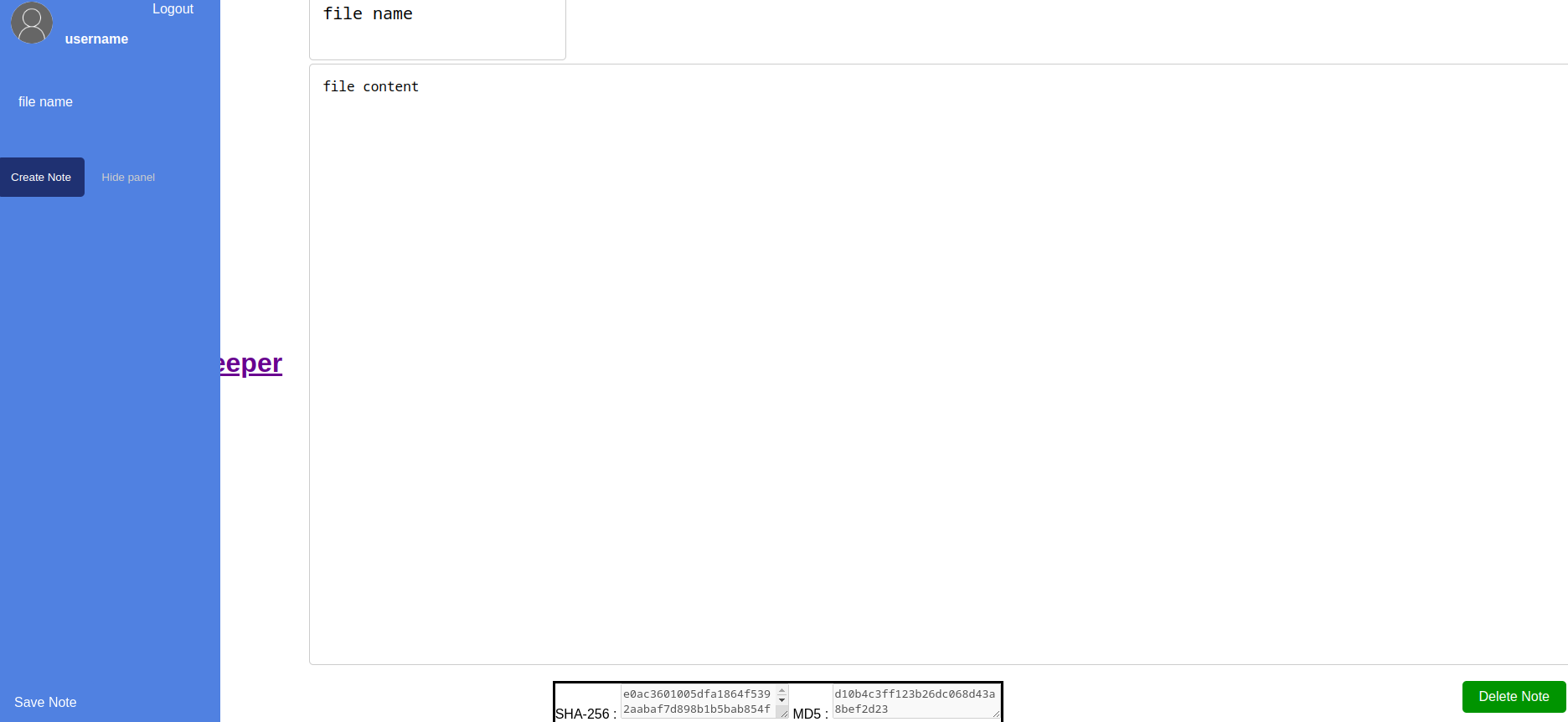
1. **About Us Page**: the page that has information about the creates and developer of this project



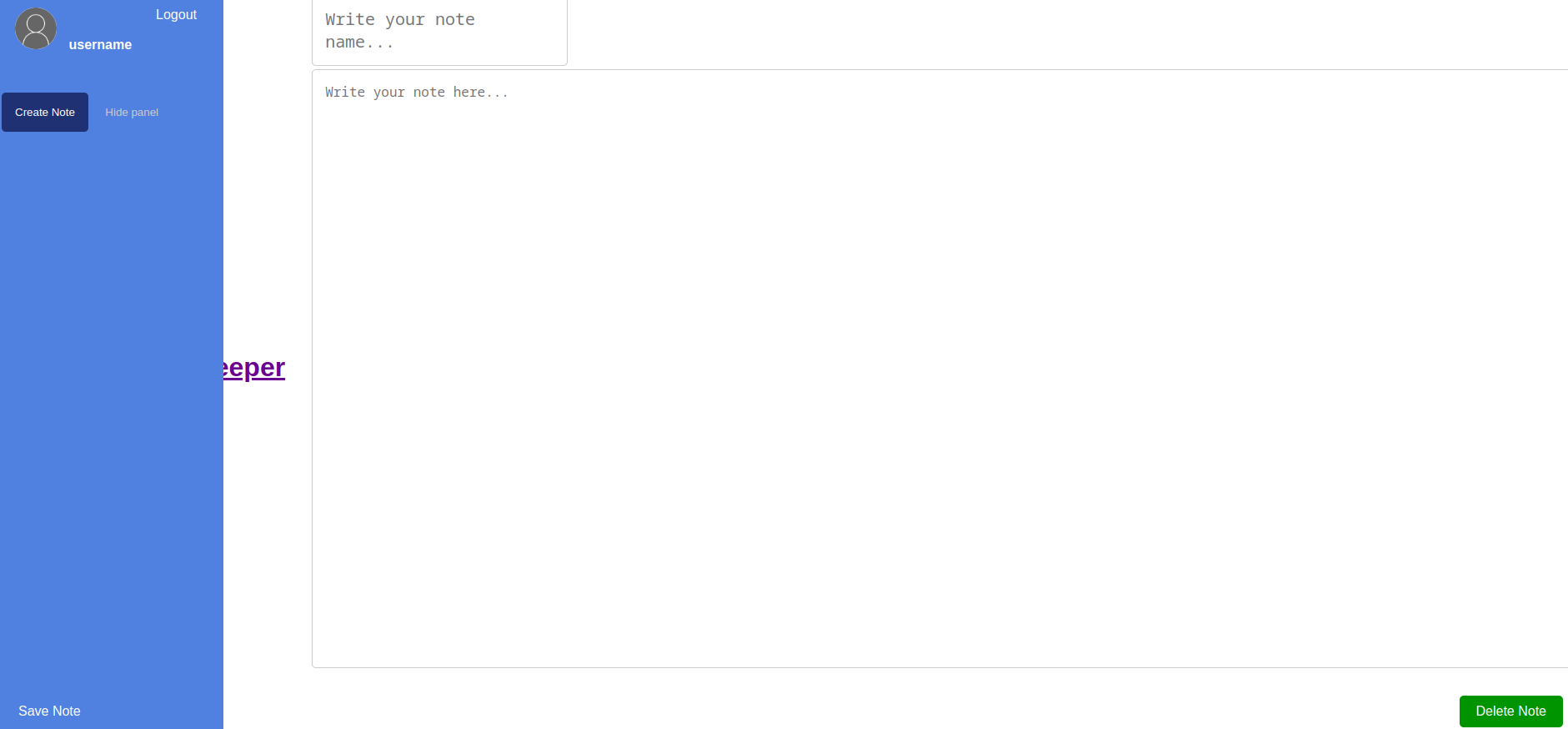
1. **Home Page**: is the one that used for creation new file, delete the existing file, edit it and view it with the hash of the content.

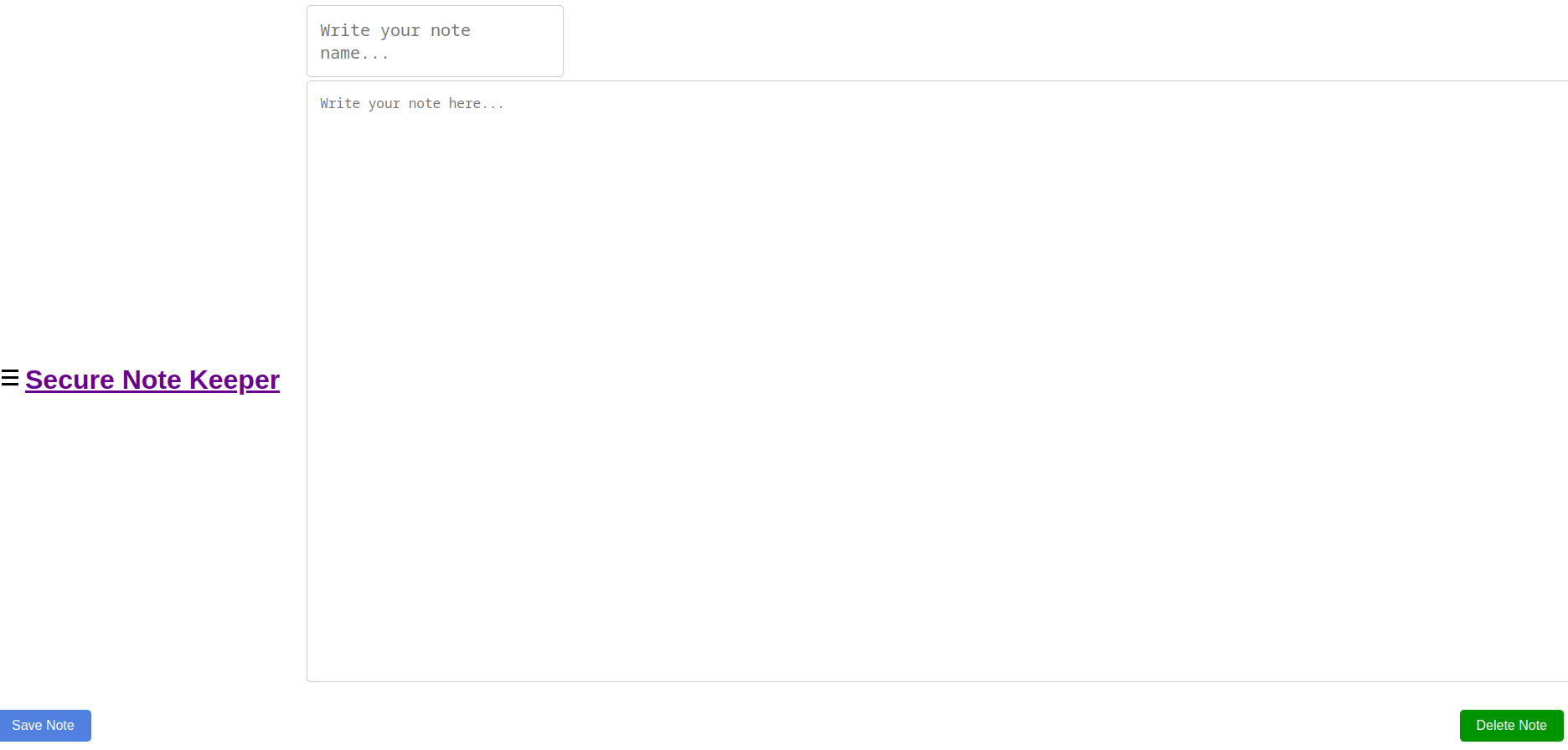
#### Secure note display



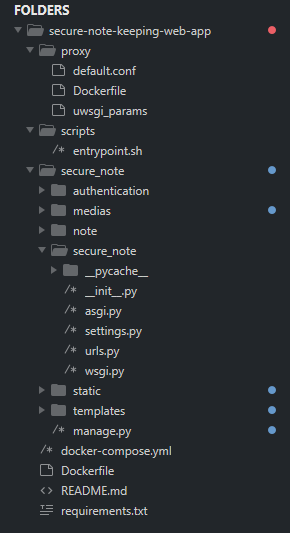
1. Secure note display with user profile

#### *Secure note create with profile*



1. Create note home page 

# **How the code is organized**

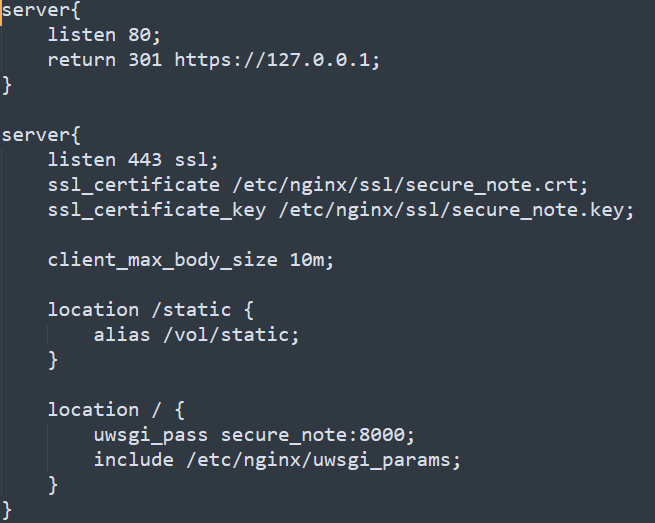


## **Proxy**

### 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

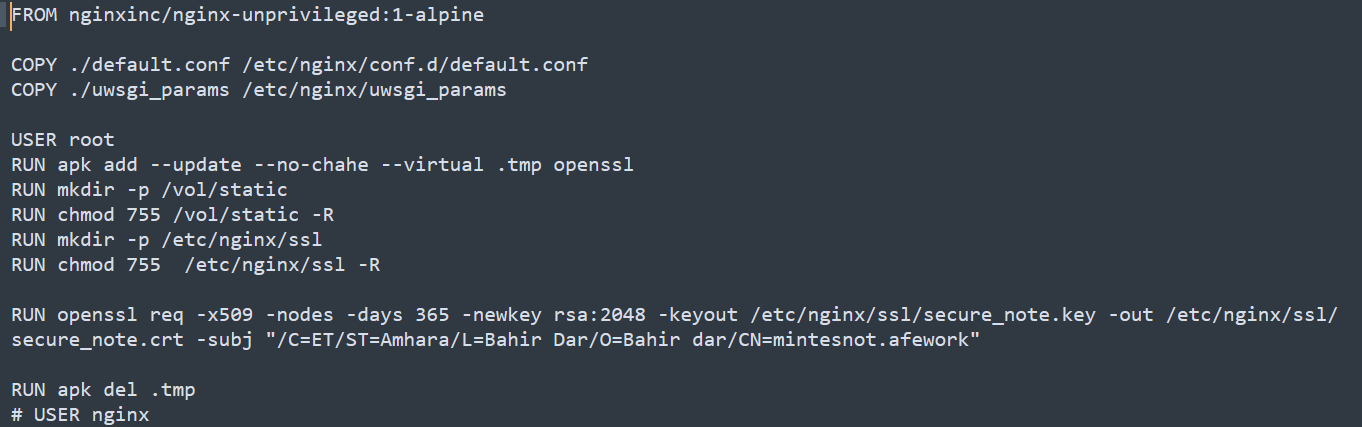
1. **The port the server listens on which are port 80 and 443** but 80 will only use to make backward compatibility and permanently redirect user https
2. **Basic configuration of self-signed ssl certificate** : In this project we have encrypted the traffic between nginx server and the client using the self-signed certificate which was generated using the openssl tool
3. 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.
4. **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
5. **The location of the Django server** to redirect the http request to Django using the uwsgi middleware



### 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 light weight as possible.



### 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:

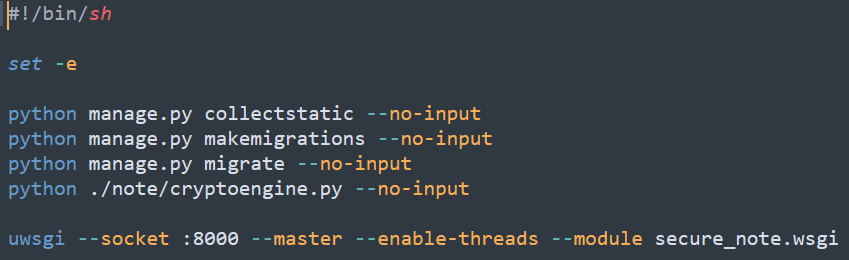
* A web client makes a request to a web server (like nginx).
* The web server reads the uwsgi\_params to determine how to communicate with the uWSGI server.
* The web server sends the request to the uWSGI server using the uwsgi protocol.
* uWSGI processes the request using your Django application, generates 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.

## **Scripts**

### 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 webserver to send request at port 8000.



## **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.

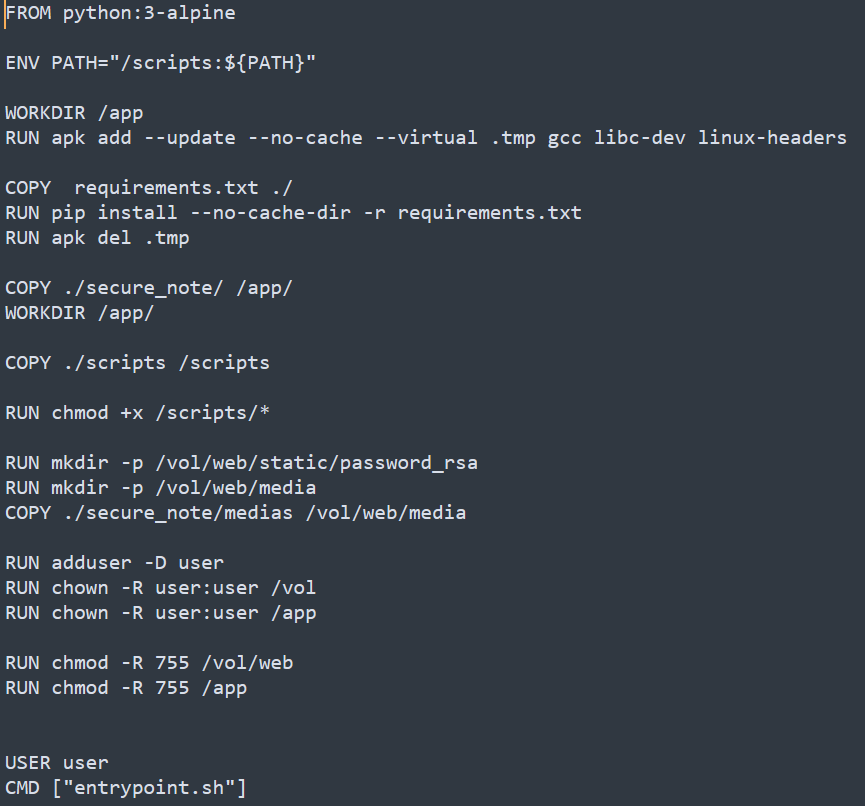


The second one is the postgres image which the image type we are going to use because of no need for additional configuration to the image we do not have any docker file. But it has environmental variables which are defined on the official postgres image documentation.

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.

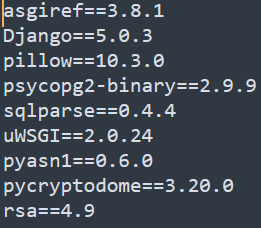
Finally we define the volumes used and need to be created locally for persistence of data. To view this we can use the docker volume ls command if we build and run the docker engine we will see this two static volume files may be with some name difference.

## **Dockerfile**



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.

## **requirments.txt**



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.

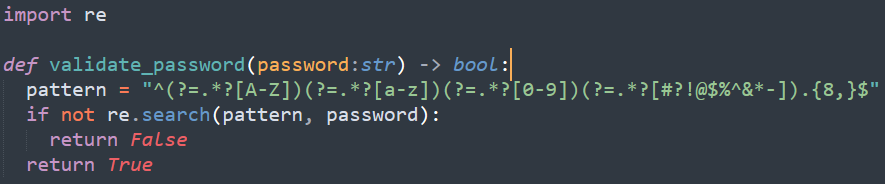
## **Secure note Django application**

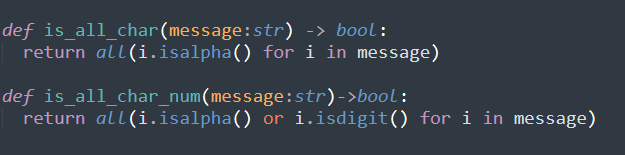
In this documentation we will try to look at the basic files and folders found in this project like the settings.py, urls.py, forms.py, models.py and views.py. For this modules we will give a detail description but for others we will have a brief explanation. We will cover modules that we design specifically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

### 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 on is found in the note application and is called crytpoengine.py.

#### Validation.py





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

#### 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.

##### MessageDigist



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.

##### RSA

###### Key generation



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

###### Encryption and Decryption

###### 

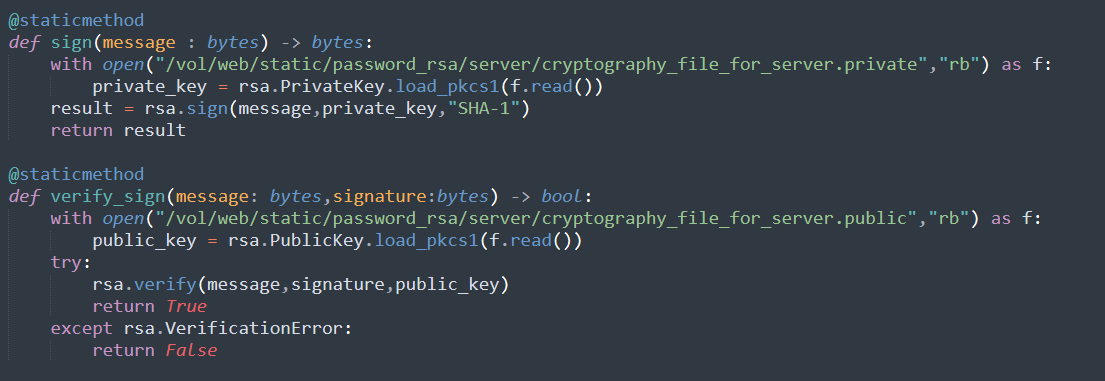
This is the cryptographic implementation of RSA to encrypt and decrypt using the user public and private key. As we can see first it will always accept a filename to identify the private and public folder for that specific user then it fetch the key based on the operation means for encryption we use 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

###### 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.



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

##### AES



###### 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

###### 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

###### Decryption

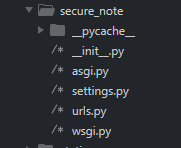
In the decryption process we have used the AES module to encrypt and let us look at it step by step:

* 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

### Django Application Flow

Our Django application contain three apps and tries to fullfil basic two fuctions which are note managnment and user authentication. The two functionalities are fulfilled by note and authientiation apps respectively. The last one or the

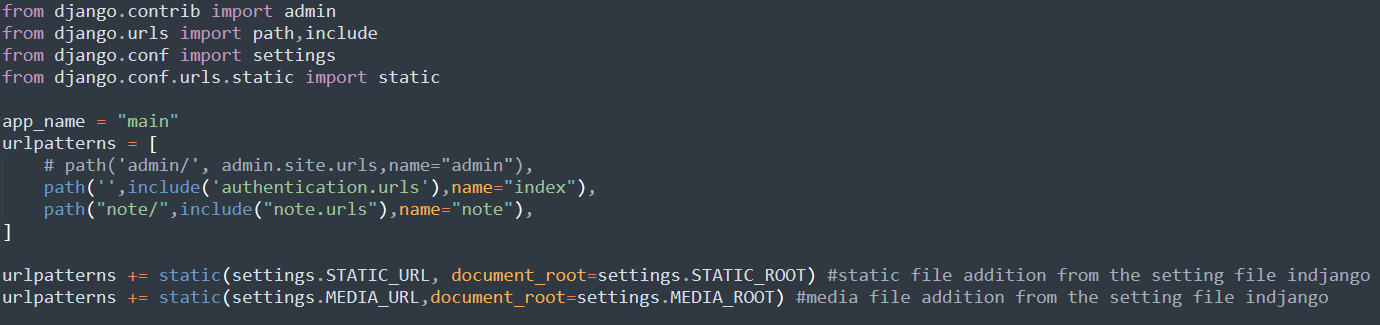
#### Secure note application



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.



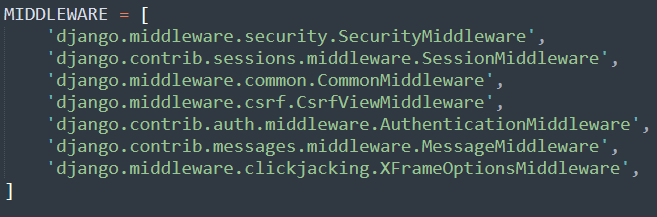
##### settings

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.

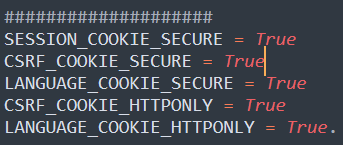


This show that it is allowed to any host to access it from any where based on the allowed host configuration

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



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



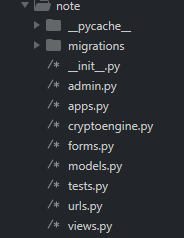
The CSRF\_COOKIE\_SECURE setting controls whether to use a secure cookie for the CSRF token. SESSION\_COOKIE\_SECURE: Set it to True to make session cookies secure.

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.

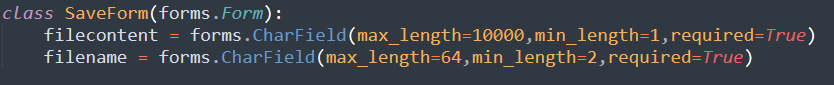
LANGUAGE\_COOKIE\_HTTPONLY: Similarly, set it to True for language cookies3.

#### Note application

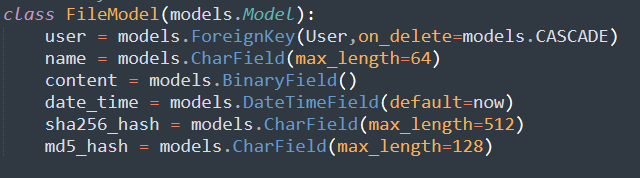


This application contain the core things that we are required to perform which are creating a note, deleteing and editing note. As we can it also contain the crypto engine module which we have discussed in the previous section on user defined part. Let us break it all and look at what they do briefly.

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.



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



Urls module contain the path used by the application to make decision on routes which begins with note. Example [https://domain/note/{anything}](https://domain/note/%7banything%7d).

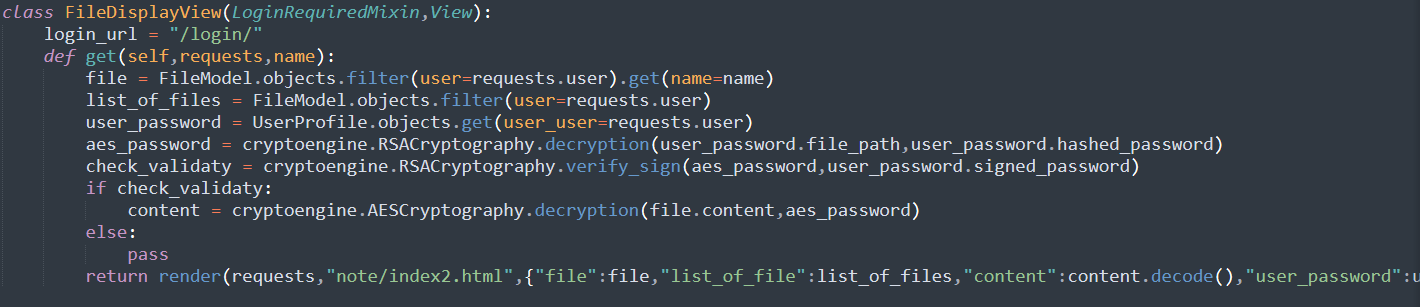
This contain the view that we have used to interact with the user and handle requests and give responses.it contain four view classes which are FileDisplay, IndexView, DeleteView and SaveFile.

##### FileDisplay

This will detch the database for the files owned y 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.

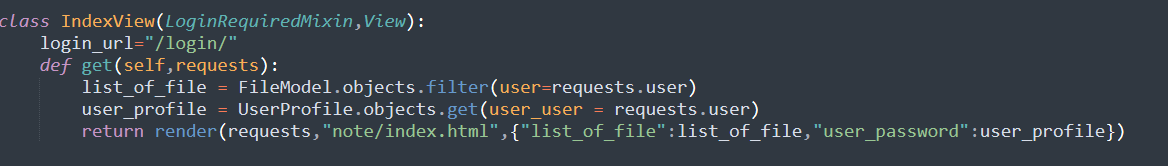
Finally it will decrypt the file content using the aes key and send it to the user.



##### 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 tmplates.



##### DeleteView



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.

##### SaveFile

#### 2,3. Authentication application

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

###### views