A FACIAL RECOGNITION-BASED AI AUTHENTICATION SYSTEM

Eren Kaya nereayak@hotmail.com

December 27, 2024

# Abstract

This is a showcase for a system by utilising Django with libraries such as Face-recognistion for creating a system to allow users to login to an app through facial recognition. The system will be able to detect the user’s face and compare it in the tables in the database to authenticate the user. The explanation on how to install and set up the evironment is in the Read Me file.

# Introduction

This application is a showcase demonstration of the usage of Django and Face-recognition libraries to create a system that allows users to login to an app through facial recognition. The system will be able to detect the user’s face and compare it to the database of faces to authenticate the user.

# Architecture

Django is used as backend framework due to its battery included philosophy. For the User Interface instead of only using Django as backend and using static for frontend, we will use Django as a full stack framework. The frontend will be created using Django’s built-in templating engine. The system will have two main pages: the login page and the profile page.

# Libraries used

the libraries can be found in the requirements.txt file. We could have used pipfiles but for the sake of simplicity we will use requirements.txt file. The libraries are as follows:

* asgiref==3.8.1
* click==8.1.8
* Django==5.1.4
* dlib==19.24.6
* face-recognition==1.3.0
* face-recognition-models==0.3.0
* numpy==2.2.1
* pillow==11.0.0
* sqlparse==0.5.3
* typing\_extensions==4.12.2

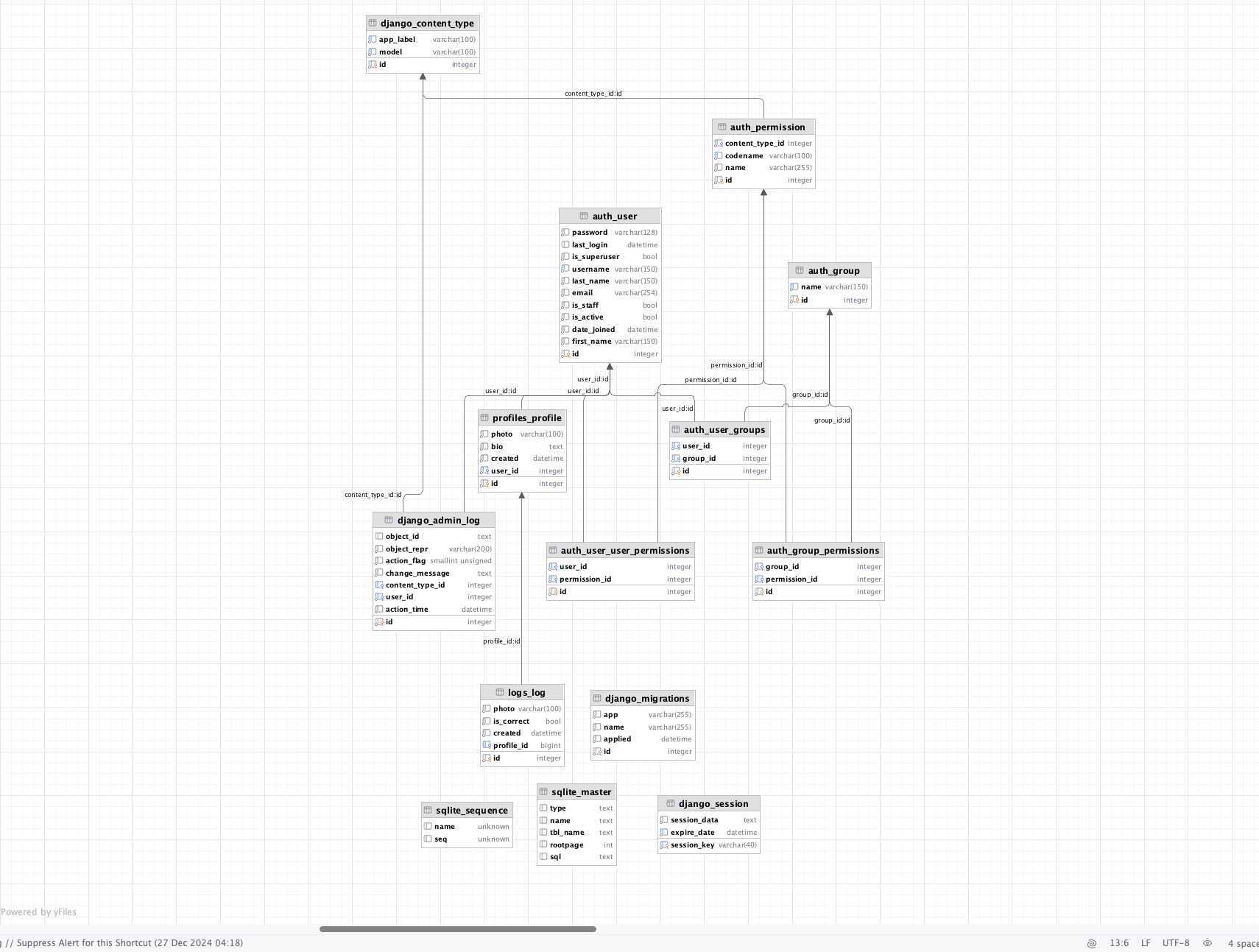
# Installation

The installation process is simple. You only need to clone the repository and fallow the steps in Readme.MD.

## Database setup

The migrations files are in migration folders of corresponding apps in the project. The migrations can be applied by running the following command:

python manage.py makemigrations python manage.py migrate

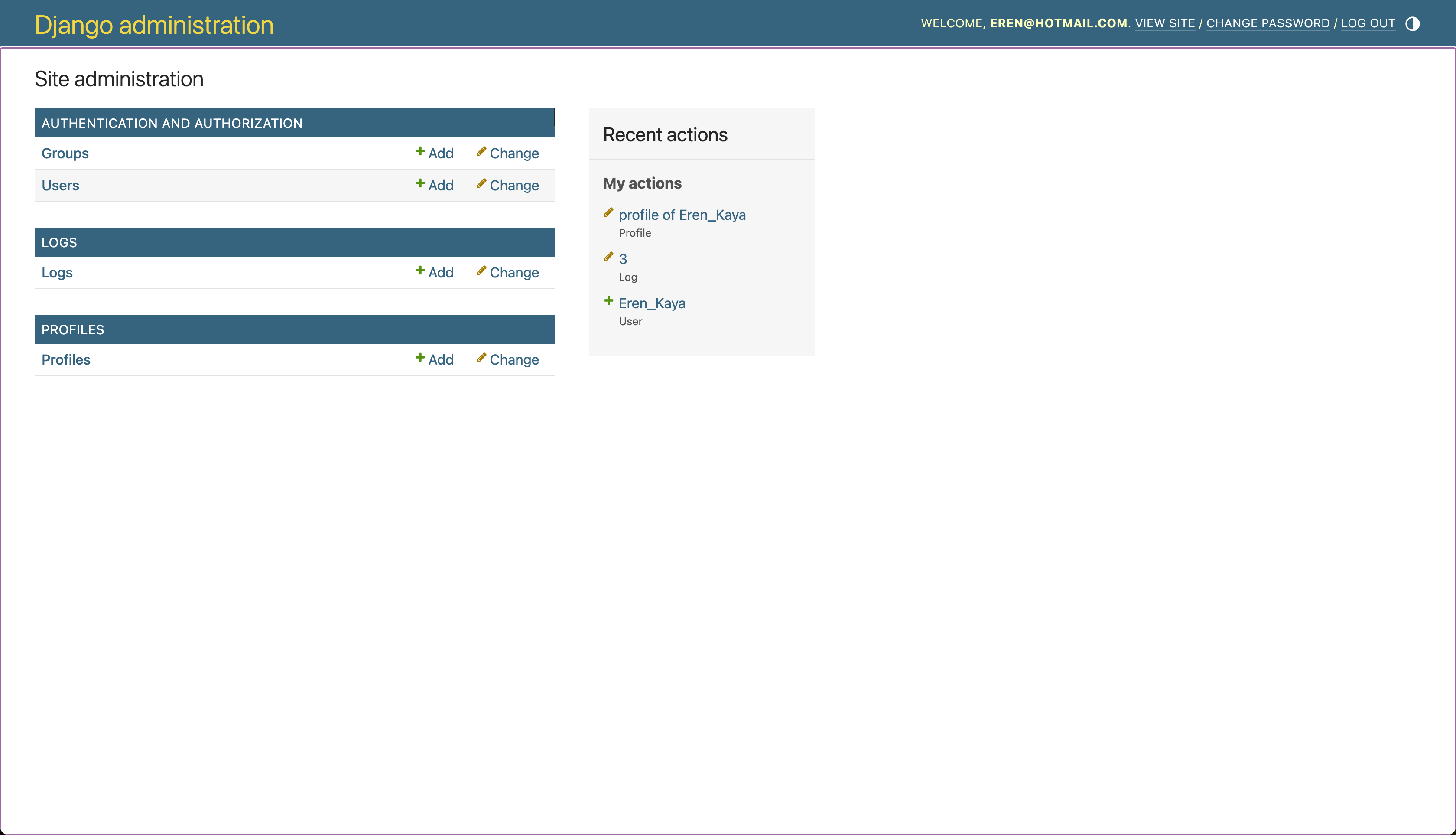


Database Structure

## Admin panel

Dajngo’s default admin panel is used for the system. The admin panel can be accessed by going to /admin and logging in with the superuser credentials. For creating super user the necessary commands are in the Make file.

python manage.py createsuperuser



Admin panel

# Created Apps, Models Structure and Signals

Profiles and Logs app is created.

For since it is for showcasing projects, the database is chosen to be SQLite database. The database is extended with two tables called Profiles and Logs.

Profiles have fallowing fields:

* user (unique one-to-one relationship with the Django User model)
* photo (image field which app stores in photos directory)
* bio (text field to store further information about the user)
* created (to keep tract of the creation time)

Logs have fallowing fields:

* profile (one-to-one relationship with the Profile model)
* photo (image field that is uploaded)
* is\_correct (boolean filed to keep track of the correctness of the system)
* created (to keep track of the creation time)

we implemented a recievers since after a user is created we need to create a profile for them. This profile will have a one-to-one relationship with the Django User model. The logs will be created after the user logs in to the system. The logs are used to keep track of the actions that come from the UI and further validate the creativeness of the system.

# UI Templates

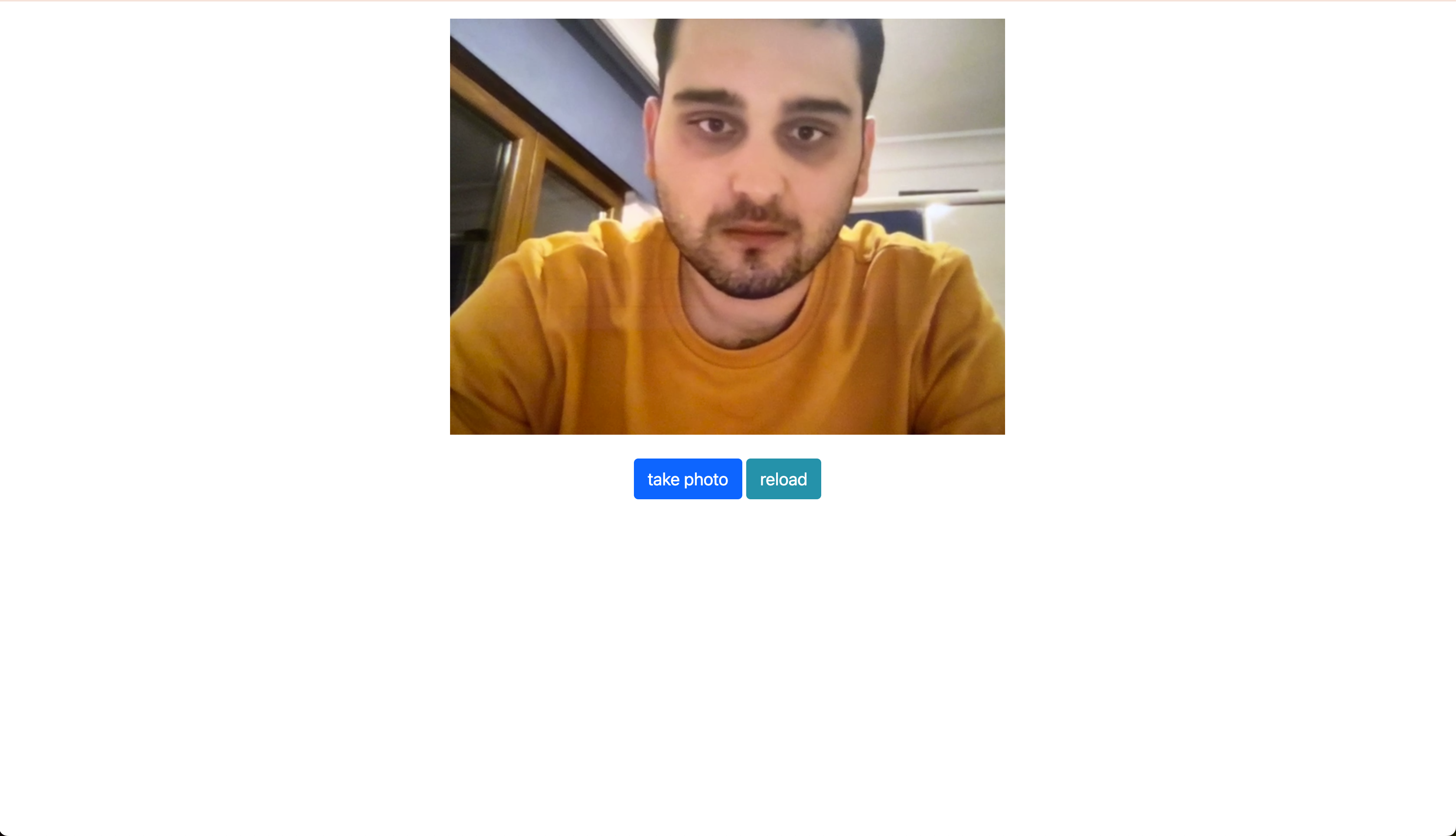
* UI templates are jinja templates that are connected to django files. The actions for creating AJAX requests are done in those files and for a good look bootstrap is used.

# Routes

MVT(model-view-template) architecture is used in this app. Thus, routes are in the view.

## Login Page:

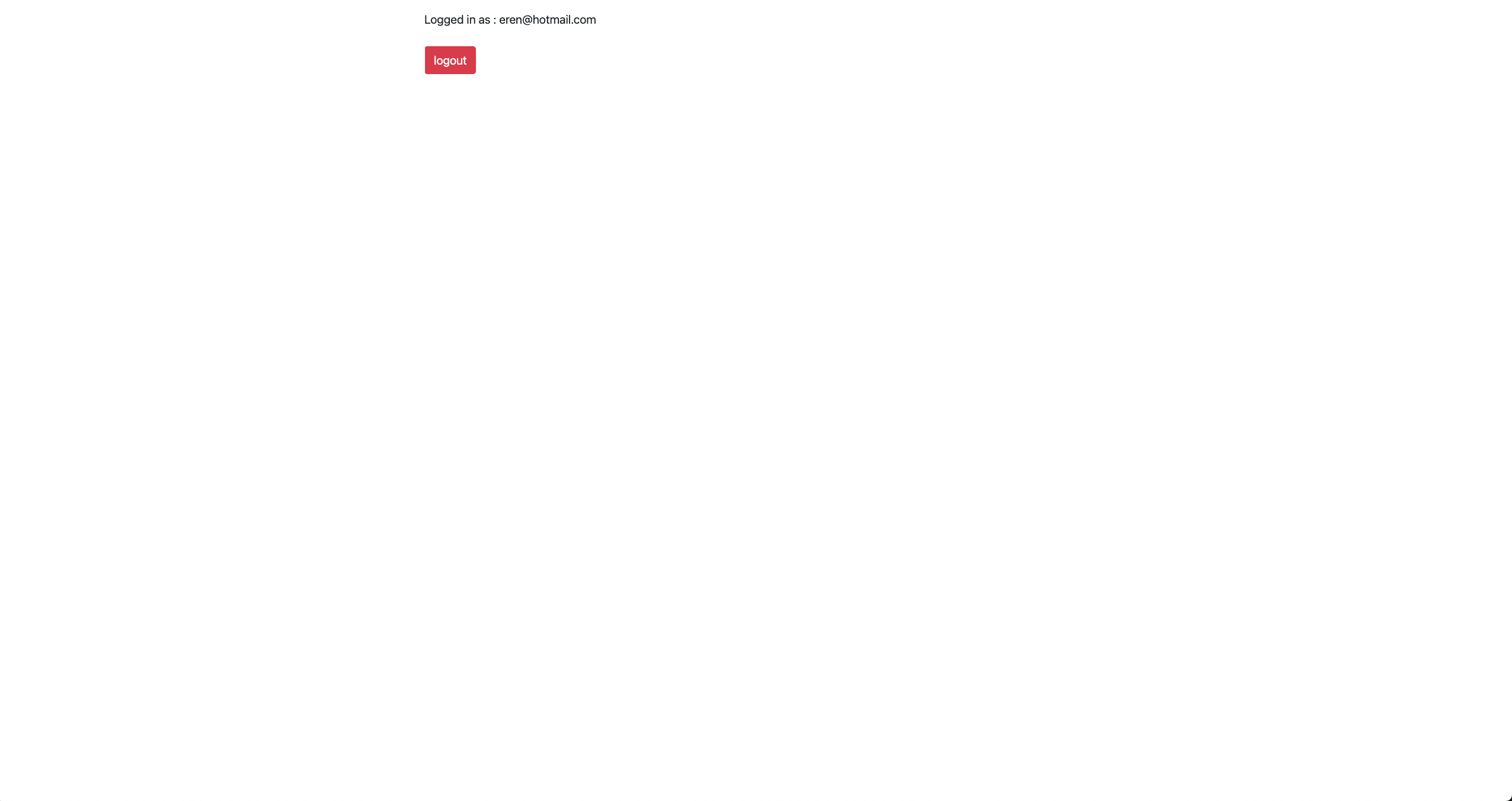
* login page is created with 'login.html' file. User can upload their picture wia their camera. Afterwards the image is sent to the backend for face recognition. If the face is recognised, the user is redirected to the home page.



Login Page

## Home Page:

* It is page where the client will be redirected after getting authenticated by the system. The page is protected with "@login\_required" decorator.



Home Page

# Functions

* we have 2 functions at the backend one for parsing the page and encode the image and the other one is for the face recognition. The face recognition function is used to compare the user’s face to the database of faces to authenticate the user.

# Security measures

CSRF token is added to the templates for security. The system is also protected against SQL injection attacks by using Django’s ORM. The system is also protected against XSS attacks by using Django’s templating engine.

# Conclusion

The app showcase the usage of an image to compare the user’s face to the database of faces to authenticate the user. The system can be improved by adding containers to deploy on kubernetes and adding extra endpoints to validate the user’s face to third party services. Since the UI will be hosted with the same domain as the backend it would not require any extra configuration for CORS. If it needed for extra authentication the authentication headers can be used to whitelist the domain in a dynamic sense. The whitelisted domains can be obscurified by accepting calls by the whitelisted ones. Overall app function well and satisfy the requirements