GRAPHICAL PASSWORD AUTHENTICATION SYSTEM

KEYWORDS:
Graphical password Authentication; Recognition techniques; Image Authentication; Information Security; Framework Architecture; Authentication

ABSTRACT
Considering the popularity and wide deployment of text passwords, we predict that they will be used as a prevalent authentication mechanism for many years to come. Thus, we have carried out studies on mechanisms to enhance text passwords. These studies suggest that password space and memorability should be improved, with an additional mechanism based on images. The combination of text and images increases resistance to some password attacks, such as brute force and observing attacks. We propose a hybrid authentication scheme integrating text and recognition-based graphical passwords. This authentication scheme can reduce the phishing attacks because if users are deceived to share their key passwords, there is still a chance to save the complete password as attackers do not know the users' image preferences. In addition to the security aspect, the proposed authentication scheme increases memorability as it does not require users to remember long and complex passwords. Thus, with the proposed scheme users will be able to create strong passwords without sacrificing usability. The hybrid scheme also offers an enjoyable sign-in/log-in experience to users.

1. Introduction

1.1 Theoretical Background

Password is a secret that is used for authentication. Passwords are the commonly used method for identifying users in computer and communication systems. It is supposed to be known only to the user. A graphical password is an authentication system that works by having the user select from images, in a specific order ,presented in a graphical user interface (GUI). For this reason, the graphical-password approach is sometimes called graphical user authentication (GUA). Human factors are often considered the weakest point in a computer security system.. Here we focus on authentication problem . User authentication is one of the important and fundamental component in most computer security systems. Biometrics is one of the important authentication methods used to tackle the problems associated with traditional username-passwords. But here we will deal with another alternative: using image as passwords. According to a recent computer world news article, the security team at a large company ran a network password cracker and within 30 seconds, they identified about 80% of the passwords. On the other hand, passwords that are difficult to guess or break are often difficult to remember. Studies showed that since user can only remember a limited number of passwords, they tend to write them down or will use the same passwords for different accounts. To address the problems with traditional username password authentication, alternative authentication methods, such as biometrics have been used.

In our project, however, we will focus on another alternative: using pictures as passwords. In addition, if the number of possible pictures is sufficiently large, the possible password space of a graphical password scheme may exceed that of text-based schemes and thus presumably offer better resistance to dictionary attacks. Because of these (presumed) advantages, there is a growing interest in graphical password.

1.2 Motivation

The main motivation behind the graphical passwords is that the people can recall or recognize graphical objects easier. It is observed that with traditional attacks it is hard to crack the graphical security systems.

1.3 Aim of the proposed Work

- Security mechanism for authentication
- Protection services against unwanted access to resources.
- Problems with Text-Based schemes
- Thus an alternative is required to:
    - 1. Increase security hence protecting your data.
    - 2. Increase the ease of accessing your data.

1.4 Objective(s) of the proposed work

- To replace the obsolete alphanumeric password strategy
- To have a strong and memorable password
- To choose a unique password to avoid being attacked.
- To help user establish a stronger password that is easy to remember yet strong at the same time.

2. Literature Survey

Wantong Zheng and Chunfu Jia proposed a method “Combined PWD”. This scheme proposes an online secret phrase verification component, combined PWD, through embedding separators (e.g., spaces) into the passwords to reinforce the current secret word validation framework. This plan uses the custom of the client’s input. In this examination, site clients can embed spaces in their secret word where they need to stop when they register a record and the site back-end records the number of spaces in each hole .

In Dec 2009 author H. Gao proposed graphical password scheme using color login. In this color login uses background color which decrease login time. Possibility of accidental login is high and password is too short. The system developed by Sobrado is improved by combining text with images or colors to generate session passwords for authentication.

Session passwords can be used only once and every time a new password is generated. The advantages of this system is that it reduces the login time, session passwords are also generated to improve security. The disadvantage of this system is that it the possibility of accidental login is high and password is too short.

In this paper M. Sreelatha proposed Hybrid Textual Authentication Scheme. This scheme uses colors and user has to rate the colors in registration phase. During login phase four pairs of colors and 8*8 matrix will be displayed. As the color rating given by the user, the password will generate. First color shows row number and second shows column number of the grid. The drawback of this system is intersecting element is the first letter of the password. The user has to memorize the rating and order of the colors. So it becomes very hectic to user. The benefit of this system is that it is flexible and simple to use.

Authors have proposed four systems which mainly focuses on graphical password. The textual-based password system is a popular authentication system since ancient times. It has many advantages but at the same time it has a few drawbacks too. Hence, the current graphical password techniques, is classify into four techniques as recognitionbased, pure recall-based, cued-recall based and hybrid based. In the recognition based algorithm the user must memorize the portfolio of images during the password creation. When the user logs in, the user must recognize the images form decoys. Various images like faces, icons, everyday objects, random arts etc. can be used.

3. Overview of the Proposed System

3.1 Introduction and Related Concepts

Graphical passwords refer to using pictures as passwords. In theory, graphical passwords are easier to remember, since human remember pictures better than words. Also they should be more resistant to brute-force attacks, force attacks, since the search space is practically infinite.

Users need to remember the password but is difficult to remember complex, random passwords. Human being has long term memory limitation, so user can forget a password that is not used regularly. Having multiple passwords may leads the user either jumble the elements of the different passwords or confuse the password for which system it corresponds to. Users normally face the password memory problems which decreases the password complexity and number of passwords. This habit reduces password security. Normally secure password should be 8 characters length, random, with upper-case characters, lowercase characters, special characters and digits.

Users ignore such password recommendations, and use instead short, simple passwords that are relatively easy to remember and easy to discover using dictionary attacks. It is often observed that users choose short password and it contain only alphabetic consisting of personal names of family or friends, pets, etc. Users generally write down their passwords as note or sometime share the passwords with others, or use the same password for multiple systems

Module for the Proposed System

A. Public Module

It is the overall viewing end of an individual website. Anyone with the URL can access this module. It is public however they can't change or alter the information.

B. User Module

The registered users are the part of user module. The user module consists of 2 functionalities - Registration and Login. During Registration, the system collects the basic details of the user like name, email, textual password, and graphical password. These all are encrypted and stored in the database. During the login phase, the user will give the username, textual password, and image password for accessing the resource. It compares the given values with data already given by the user at the registration phase. If it matched, then he/she will be logged into the page.

C. Account and Settings

This is the third module that contains the client's records and different settings of the computerized web stage. There is a link between the user module and the account module, If the user completes the registration, then the account will be created on the database. Also, the users can change their passwords at any time. Sign-in data, privacy and security choices, and so on are a benefit of it. Furthermore, clients can get warnings and request support from this part.

COMPONENTS OF THE PROJECT:

1. admin.py : We have created this file for accessing the database as the admin. In this file, we have imported details from models file which comprises of input credentials of the user and store it together at a place.
For doing this we have used modules admin for contribution module of Django and models module for receiving information from models file in the project.
From models module we have imported LoginInfo class so as to get access of the input credentials from the file. After getting the access to models file in project through model module, we have used register command so as the details get registered in the database of the admin.

2. apps.py : This file is created to include any application configuration for the app. Using this, we configure some of the attributes of the application.

3. models.py : We have made this file so as to check whether the credentials entered by the user for the first time when he/she created an account of its own is unique or not and have entered the command for comparing and checking of cases such as login fail, reset password, failure in entering the wrong password and checking whether the credentials entered initially don't cascade with the already existing credentials.In this file we have used modules such as models for Django and contrib.auth from Django which provides API reference material for the components of Django's authentication system. While using this we have used class User for letting the users with proper authentication use the site, i.e., those users whose credentials are unique from other users. If there is any kind of failure condition or resetting of password happens, then it returns back username only along with a link if needed for that case.

4. test.py : This file comprises of testcases for this project. Here there will be various situational cases if the program faces any malfunction.

5. urls.py : We have created this file so as to create urls for homepage, register page, login page, logout page, login from uid, reset view and reset from uid. It comprises of url pattern for each mentioned above along with their name which will be displayed on webpage when using it.

6. views.py : This file comprises of a total layout of the project. In this file, there are various classes such as getting password images, updating login information, sending reset password mail to user, sending login link mail to user and all the webpages. This file has the main functional commands for every command prompt or shift of webpage, which makes it the skeleton of the code.

7. manage.py : This file comprises of commands for winding up the total project in one unit and while executing it, this file makes all the files linked up together into one framework. This has os and sys module imported which helps in clubbing up various framework into one unit.

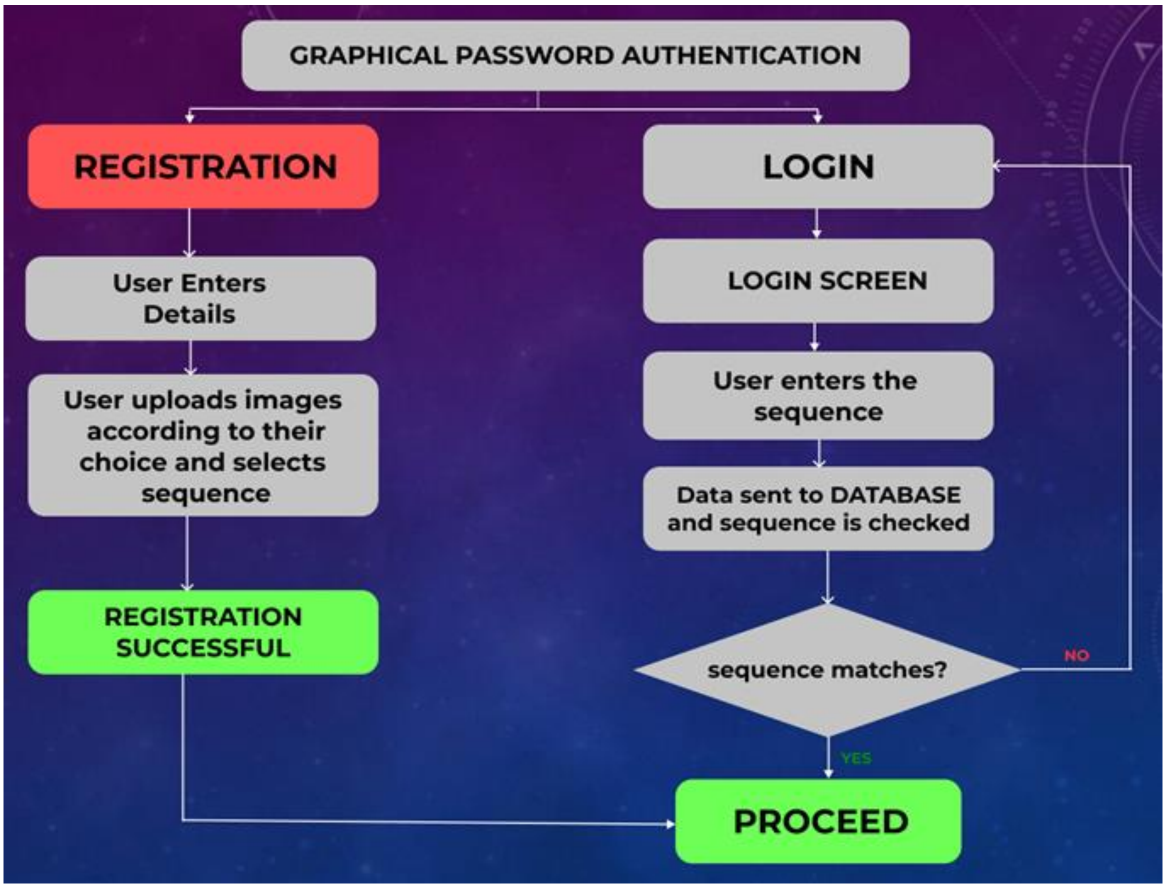
8. HTML files: There are various HTML files which frame the webpage and add colour and look to this project. Along with adding colour and look, this also comprises of links for shifting from one webpage to another, whose urls were created in url file.

PROBLEM STATEMENT

Password is a secret that is used for authentication. Passwords are the commonly used method for identifying users in computer and communication systems. It is supposed to be known only to the user. A graphical password is an authentication system that works by having the user select from images, in a specific order ,presented in a graphical user interface (GUI). For this reason, the graphical-password approach is sometimes called graphical user authentication (GUA). Human factors are often considered the weakest point in a computer security system.. Here we focus on authentication problem . User authentication is one of the important and fundamental component in most computer security systems. Biometrics is one of the important authentication methods used to tackle the problems associated with traditional username-passwords. But here we will deal with another alternative: using image as passwords. According to a recent computer world news article, the security team at a large company ran a network password cracker and within 30 seconds, they identified about 80% of the passwords. On the other hand, passwords that are difficult to guess or break are often difficult to remember. Studies showed that since user can only remember a limited number of passwords, they tend to write them down or will use the same passwords for different accounts. To address the problems with traditional username password authentication, alternative authentication methods, such as biometrics have been used.

In our project, however, we will focus on another alternative: using pictures as passwords. In addition, if the number of possible pictures is sufficiently large, the possible password space of a graphical password scheme may exceed that of text-based schemes and thus presumably offer better resistance to dictionary attacks. Because of these (presumed) advantages, there is a growing interest in graphical password.

<u>ARCHITECTURE DIAGRAM</u>


GRAPHICAL PASSWORD AUTHENTICATION
REGISTRATION
User Enters Details
User uploads images according to their choice and selects sequence
REGISTRATION SUCCESSFUL
LOGIN
LOGIN SCREEN
User enters the sequence
Data sent to DATABASE and sequence is checked
sequence matches?
NO
YES
PROCEED

Algorithm:

```
1       Start
2       home→Register→Enter username→Enter email
3       print random images to prevent attack
        images = random.sample()
        print(images)
4       select images as password
5       login_info = loginInfo(user = user, fails = 0)
        login_info.save()
        messages.success('Account created!')
        return to home
        except Exception:
        messages.warning('Error!')
6       Redirect to login

7       Login→Enter username→Enter password
8       If didSuccess == true
        User.logininfo.fails = 0
        messages.success('Login Successful!')
                Else
                        User.logininfo.fails += 1
                        messages.warning('incorrect credentials!')
                        Print('failed attempts: '.format(user.username,
            user.logininfo.fails))
                Redirect to login
9       if user.logininfo.fails >= TBA
        print('isBlocked: ' .format(userlogininfo, TBA))
10      send email only if user.logininfo.login_link is None
        if user.logininfo.login_link is None
        link = str(uuid.uuid4())
        user.logininfo.login_link = link
        user.logininfo.save()
```

```
11   email = EmailMessage(subject = ‘link to login to you account.’,
     body = ‘’’ --- someone tried to brute force your account ---
     link: http://{}8000/login/{} --- ‘’’
     .format(ALLOWED_HOST[-1], link), from_email = EMAIL_HOST_USER,
     to = [user.email])
     Email.send()
12   send reset link every time user requests
13   repeat 10
14   email = EmailMessage(subject = ‘link to reset your password.’,
     body = ‘’’ --- you have requested to reset your password ---
     link: http://{}8000/login/{} --- ‘’’
     .format(ALLOWED_HOST[-1], link), from_email = EMAIL_HOST_USER,
     to = [user.email])
     Email.send()
15   if logout(request)
     messages.warning(‘you’ve been logged out!’)
16   redirect to home
17   end
```

SYSTEM REQUIREMENTS

H/W Requirements

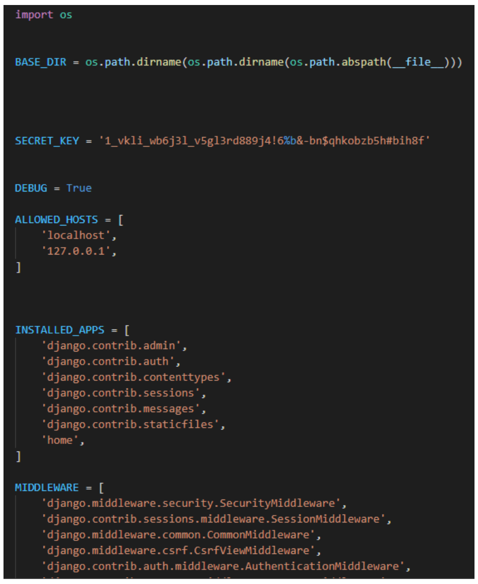
- A standard processor
- RAM: Minimum 1 GB.
- Standard Keyboard and Mouse.

S/W Requirements

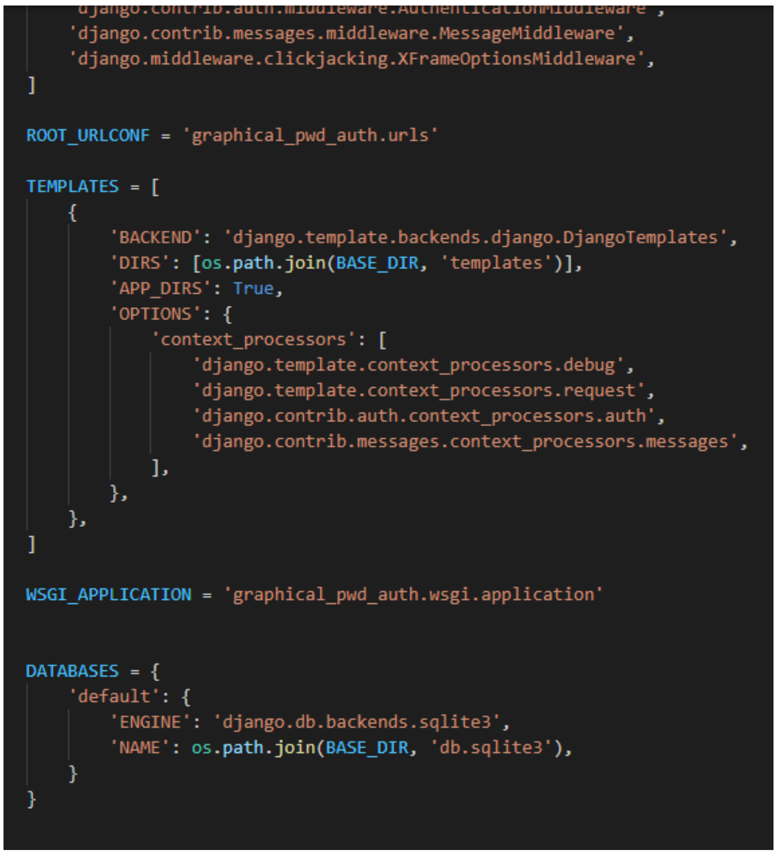
- Operating System: Windows 7 and above
- Visual studio code
- Django

CODE SS

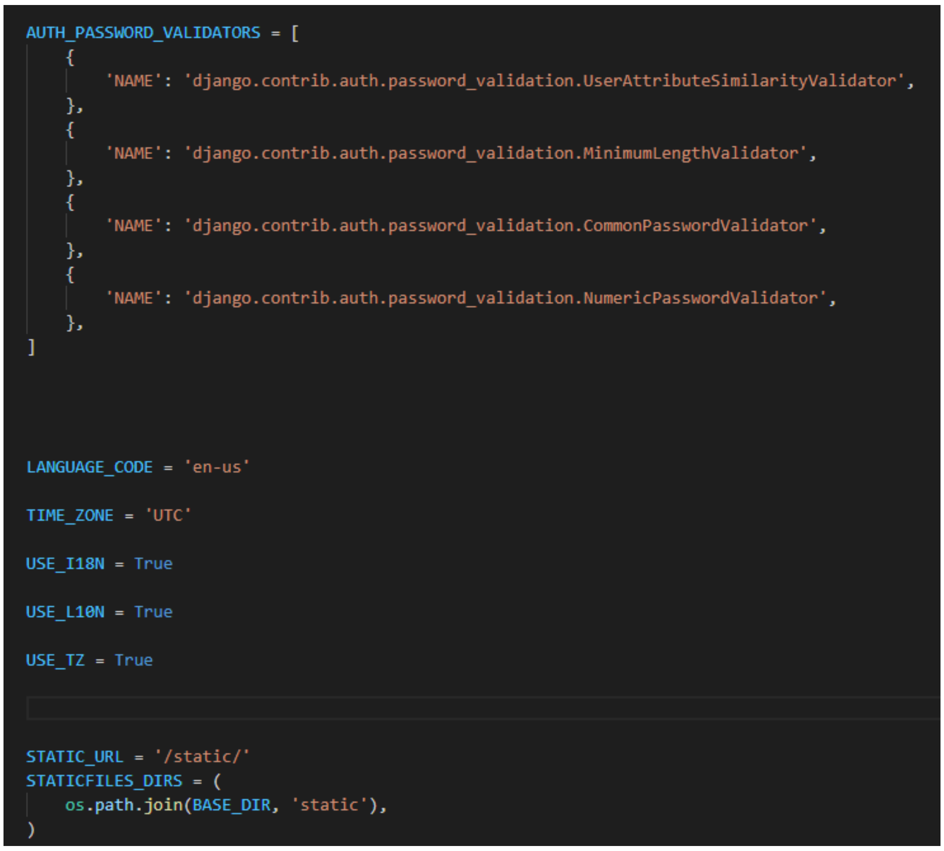
1.settings.py

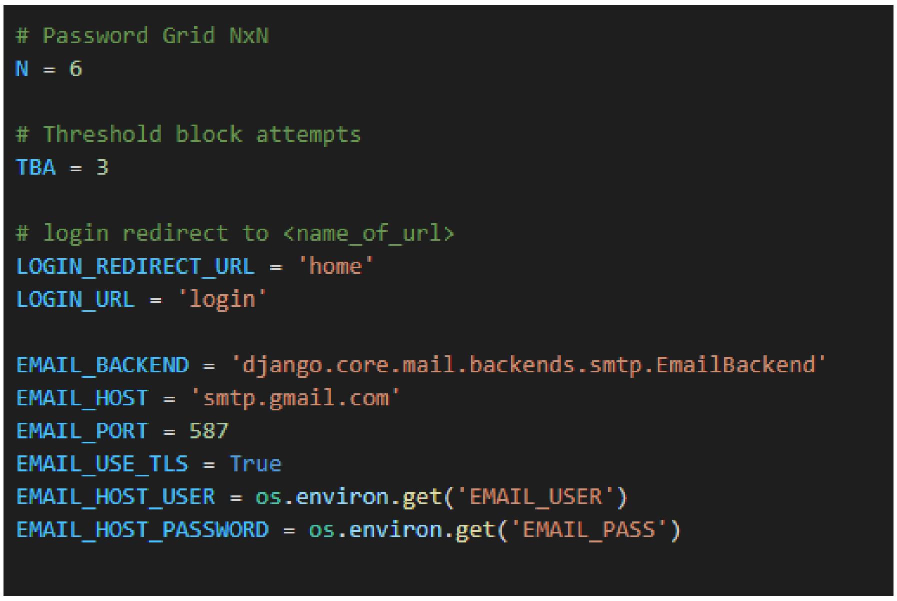


```python
import os


BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))




SECRET_KEY = '1_vkli_wb6j3l_v5gl3rd889j4!6%b&-bn$qhkobzb5h#bih8f'


DEBUG = True

ALLOWED_HOSTS = [
    'localhost',
    '127.0.0.1',
]



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

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
```

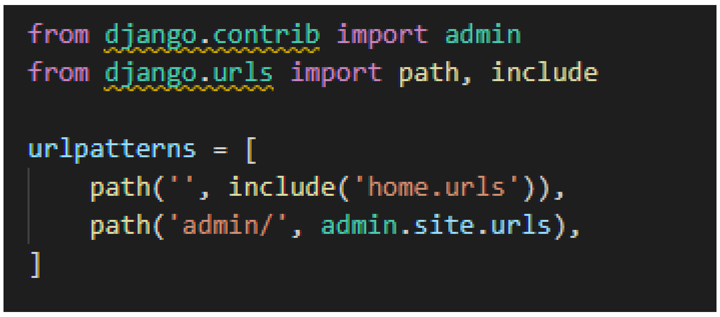


```python
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'graphical_pwd_auth.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'graphical_pwd_auth.wsgi.application'


DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

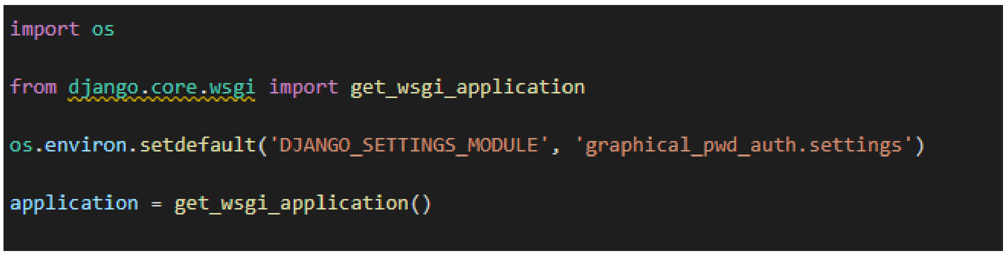

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]



LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True



STATIC_URL = '/static/'
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'static'),
)
```


```
# Password Grid NxN
N = 6

# Threshold block attempts
TBA = 3

# login redirect to <name_of_url>
LOGIN_REDIRECT_URL = 'home'
LOGIN_URL = 'login'

EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = os.environ.get('EMAIL_USER')
EMAIL_HOST_PASSWORD = os.environ.get('EMAIL_PASS')
```

2.urls.py

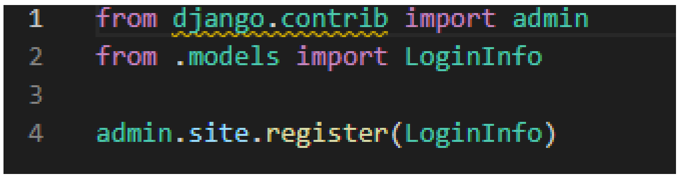

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('', include('home.urls')),
    path('admin/', admin.site.urls),
]
```

3.wsgi.py



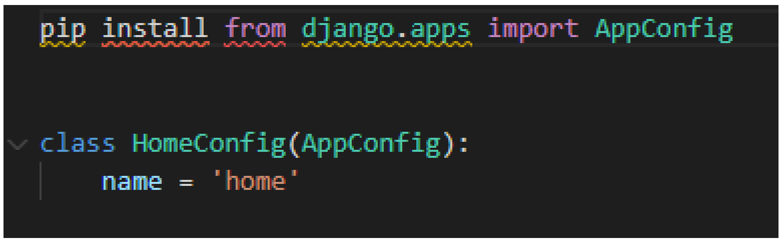
```python
import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'graphical_pwd_auth.settings')

application = get_wsgi_application()
```

4.admin.py



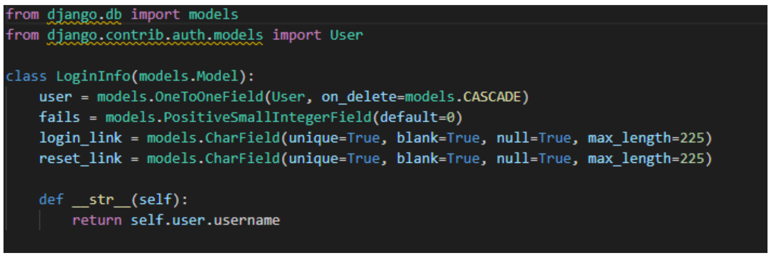
```python
from django.contrib import admin
from .models import LoginInfo

admin.site.register(LoginInfo)
```

5.apps.py



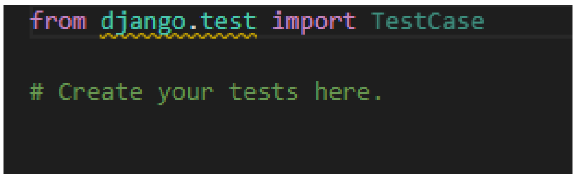
```python
pip install from django.apps import AppConfig


class HomeConfig(AppConfig):
    name = 'home'
```

6.models.py



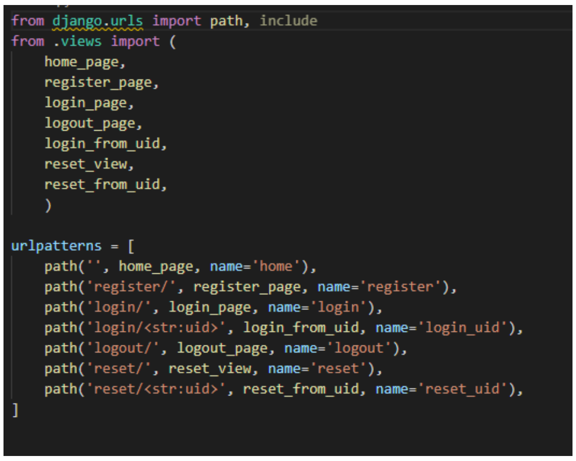
```python
from django.db import models
from django.contrib.auth.models import User

class LoginInfo(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    fails = models.PositiveSmallIntegerField(default=0)
    login_link = models.CharField(unique=True, blank=True, null=True, max_length=225)
    reset_link = models.CharField(unique=True, blank=True, null=True, max_length=225)

    def __str__(self):
        return self.user.username
```

7.tests.py



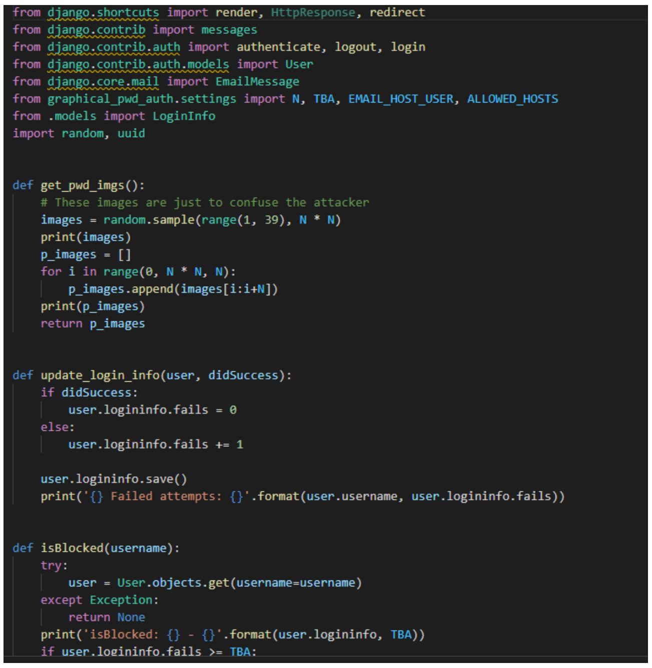
```python
from django.test import TestCase

# Create your tests here.
```

8.urls.py

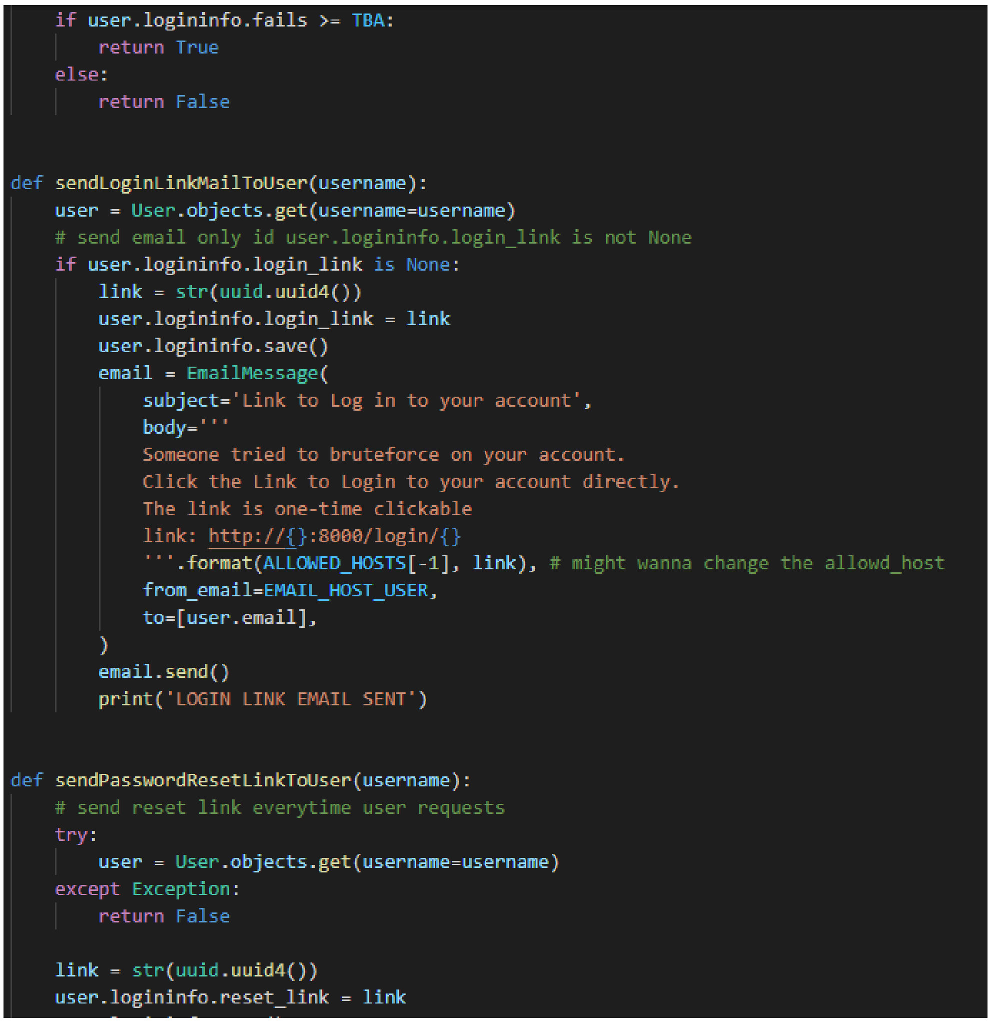


```python
from django.urls import path, include
from .views import (
    home_page,
    register_page,
    login_page,
    logout_page,
    login_from_uid,
    reset_view,
    reset_from_uid,
    )

urlpatterns = [
    path('', home_page, name='home'),
    path('register/', register_page, name='register'),
    path('login/', login_page, name='login'),
    path('login/<str:uid>', login_from_uid, name='login_uid'),
    path('logout/', logout_page, name='logout'),
    path('reset/', reset_view, name='reset'),
    path('reset/<str:uid>', reset_from_uid, name='reset_uid'),
]
```

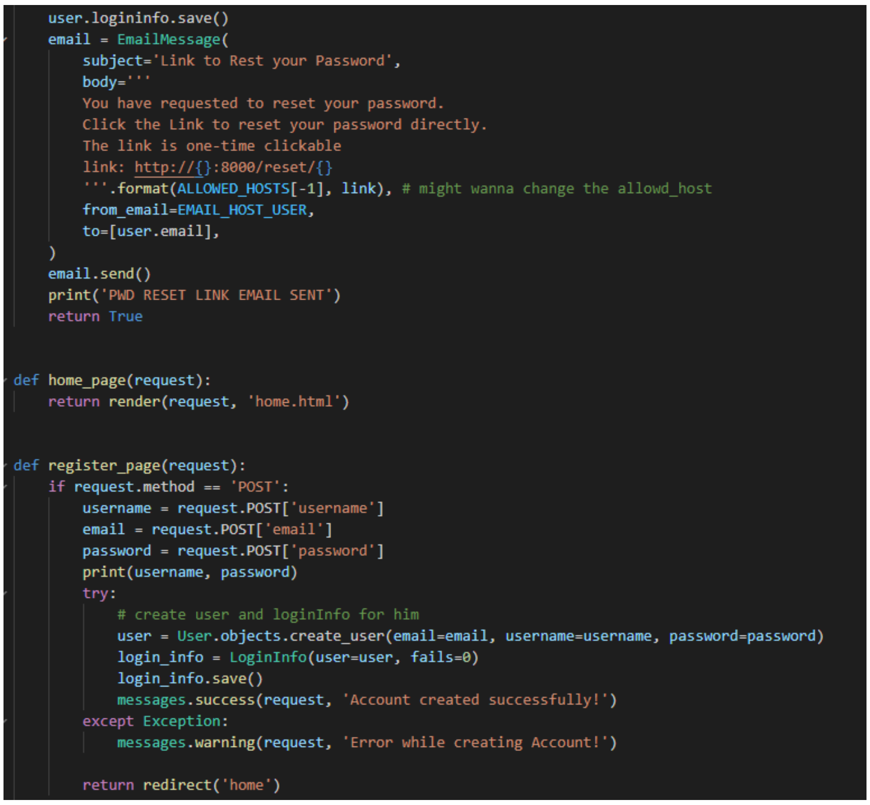
9.views.py



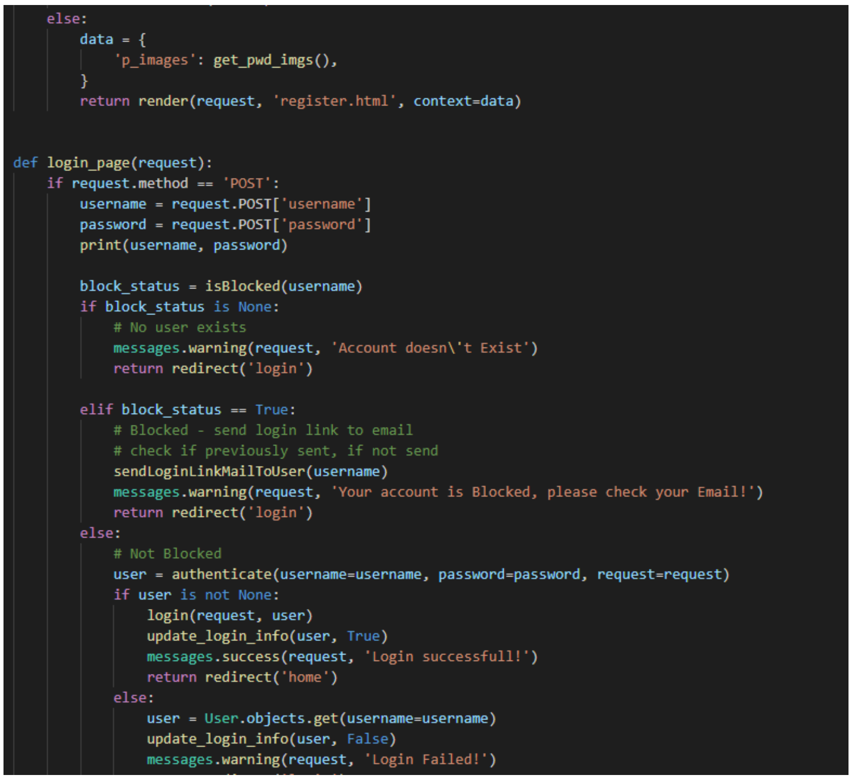
```python
from django.shortcuts import render, HttpResponse, redirect
from django.contrib import messages
from django.contrib.auth import authenticate, logout, login
from django.contrib.auth.models import User
from django.core.mail import EmailMessage
from graphical_pwd_auth.settings import N, TBA, EMAIL_HOST_USER, ALLOWED_HOSTS
from .models import LoginInfo
import random, uuid


def get_pwd_imgs():
    # These images are just to confuse the attacker
    images = random.sample(range(1, 39), N * N)
    print(images)
    p_images = []
    for i in range(0, N * N, N):
        p_images.append(images[i:i+N])
    print(p_images)
    return p_images


def update_login_info(user, didSuccess):
    if didSuccess:
        user.logininfo.fails = 0
    else:
        user.logininfo.fails += 1

    user.logininfo.save()
    print('{} Failed attempts: {}'.format(user.username, user.logininfo.fails))


def isBlocked(username):
    try:
        user = User.objects.get(username=username)
    except Exception:
        return None
    print('isBlocked: {} - {}'.format(user.logininfo, TBA))
    if user.logininfo.fails >= TBA:
```



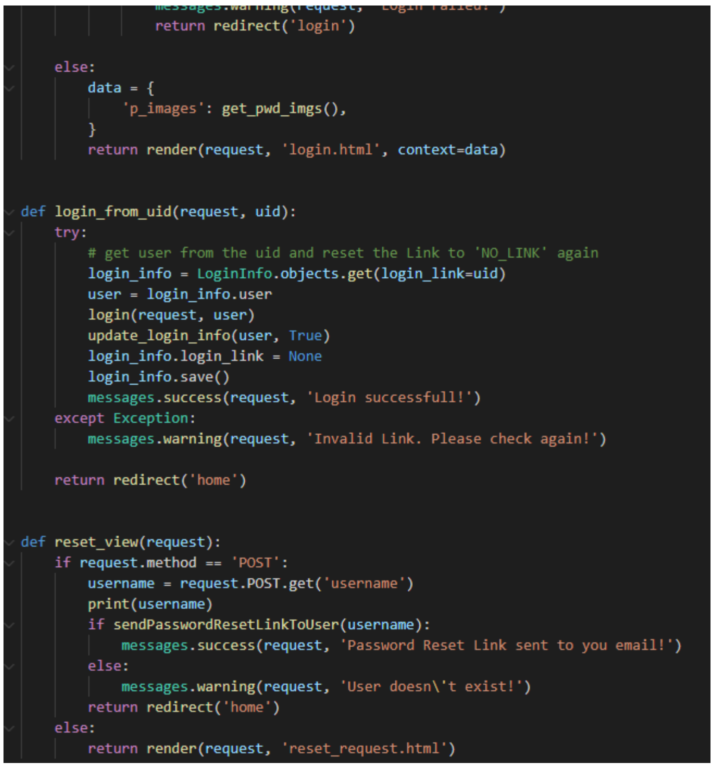
```python
    if user.logininfo.fails >= TBA:
        return True
    else:
        return False


def sendLoginLinkMailToUser(username):
    user = User.objects.get(username=username)
    # send email only id user.logininfo.login_link is not None
    if user.logininfo.login_link is None:
        link = str(uuid.uuid4())
        user.logininfo.login_link = link
        user.logininfo.save()
        email = EmailMessage(
            subject='Link to Log in to your account',
            body='''
            Someone tried to bruteforce on your account.
            Click the Link to Login to your account directly.
            The link is one-time clickable
            link: http://{}:8000/login/{}
            '''.format(ALLOWED_HOSTS[-1], link), # might wanna change the allowd_host
            from_email=EMAIL_HOST_USER,
            to=[user.email],
        )
        email.send()
        print('LOGIN LINK EMAIL SENT')


def sendPasswordResetLinkToUser(username):
    # send reset link everytime user requests
    try:
        user = User.objects.get(username=username)
    except Exception:
        return False

    link = str(uuid.uuid4())
    user.logininfo.reset_link = link
```



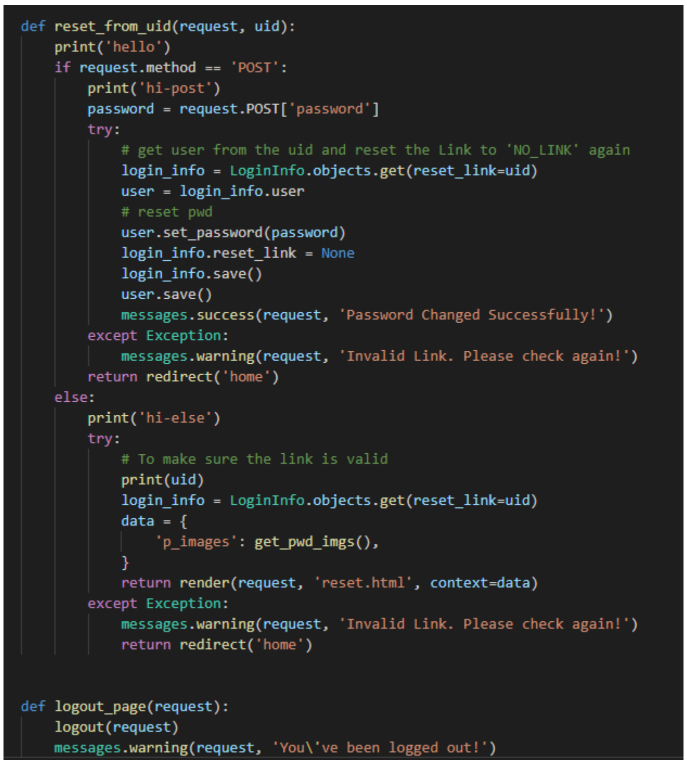
```python
    user.logininfo.save()
    email = EmailMessage(
        subject='Link to Rest your Password',
        body='''
        You have requested to reset your password.
        Click the Link to reset your password directly.
        The link is one-time clickable
        link: http://{}:8000/reset/{}
        '''.format(ALLOWED_HOSTS[-1], link), # might wanna change the allowd_host
        from_email=EMAIL_HOST_USER,
        to=[user.email],
    )
    email.send()
    print('PWD RESET LINK EMAIL SENT')
    return True


def home_page(request):
    return render(request, 'home.html')


def register_page(request):
    if request.method == 'POST':
        username = request.POST['username']
        email = request.POST['email']
        password = request.POST['password']
        print(username, password)
        try:
            # create user and loginInfo for him
            user = User.objects.create_user(email=email, username=username, password=password)
            login_info = LoginInfo(user=user, fails=0)
            login_info.save()
            messages.success(request, 'Account created successfully!')
        except Exception:
            messages.warning(request, 'Error while creating Account!')

        return redirect('home')
```



```python
    else:
        data = {
            'p_images': get_pwd_imgs(),
        }
        return render(request, 'register.html', context=data)


def login_page(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        print(username, password)

        block_status = isBlocked(username)
        if block_status is None:
            # No user exists
            messages.warning(request, 'Account doesn\'t Exist')
            return redirect('login')

        elif block_status == True:
            # Blocked - send login link to email
            # check if previously sent, if not send
            sendLoginLinkMailToUser(username)
            messages.warning(request, 'Your account is Blocked, please check your Email!')
            return redirect('login')
        else:
            # Not Blocked
            user = authenticate(username=username, password=password, request=request)
            if user is not None:
                login(request, user)
                update_login_info(user, True)
                messages.success(request, 'Login successfull!')
                return redirect('home')
            else:
                user = User.objects.get(username=username)
                update_login_info(user, False)
                messages.warning(request, 'Login Failed!')
```

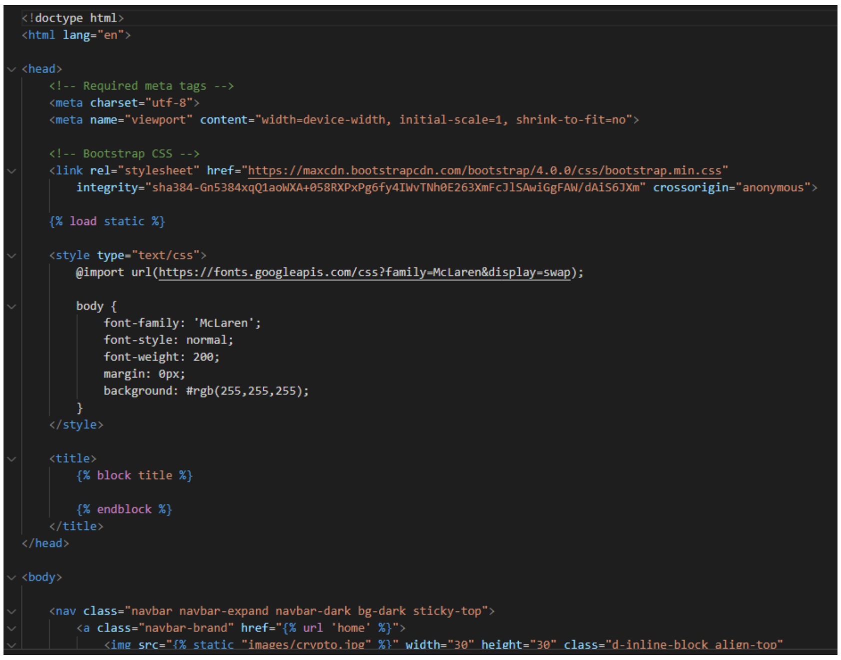


```python
                return redirect('login')

    else:
        data = {
            'p_images': get_pwd_imgs(),
        }
        return render(request, 'login.html', context=data)


def login_from_uid(request, uid):
    try:
        # get user from the uid and reset the Link to 'NO_LINK' again
        login_info = LoginInfo.objects.get(login_link=uid)
        user = login_info.user
        login(request, user)
        update_login_info(user, True)
        login_info.login_link = None
        login_info.save()
        messages.success(request, 'Login successfull!')
    except Exception:
        messages.warning(request, 'Invalid Link. Please check again!')

    return redirect('home')


def reset_view(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        print(username)
        if sendPasswordResetLinkToUser(username):
            messages.success(request, 'Password Reset Link sent to you email!')
        else:
            messages.warning(request, 'User doesn\'t exist!')
        return redirect('home')
    else:
        return render(request, 'reset_request.html')
```

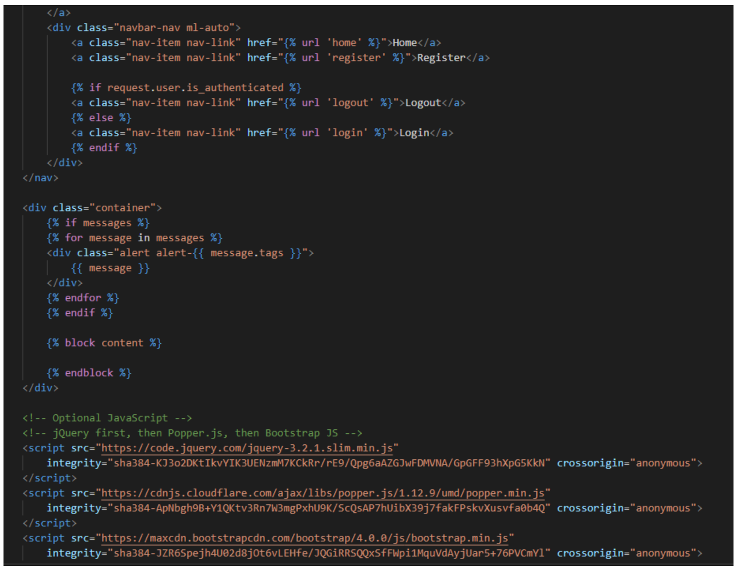


```python
def reset_from_uid(request, uid):
    print('hello')
    if request.method == 'POST':
        print('hi-post')
        password = request.POST['password']
        try:
            # get user from the uid and reset the Link to 'NO_LINK' again
            login_info = LoginInfo.objects.get(reset_link=uid)
            user = login_info.user
            # reset pwd
            user.set_password(password)
            login_info.reset_link = None
            login_info.save()
            user.save()
            messages.success(request, 'Password Changed Successfully!')
        except Exception:
            messages.warning(request, 'Invalid Link. Please check again!')
        return redirect('home')
    else:
        print('hi-else')
        try:
            # To make sure the link is valid
            print(uid)
            login_info = LoginInfo.objects.get(reset_link=uid)
            data = {
                'p_images': get_pwd_imgs(),
            }
            return render(request, 'reset.html', context=data)
        except Exception:
            messages.warning(request, 'Invalid Link. Please check again!')
            return redirect('home')


def logout_page(request):
    logout(request)
    messages.warning(request, 'You\'ve been logged out!')
```

10.base.html

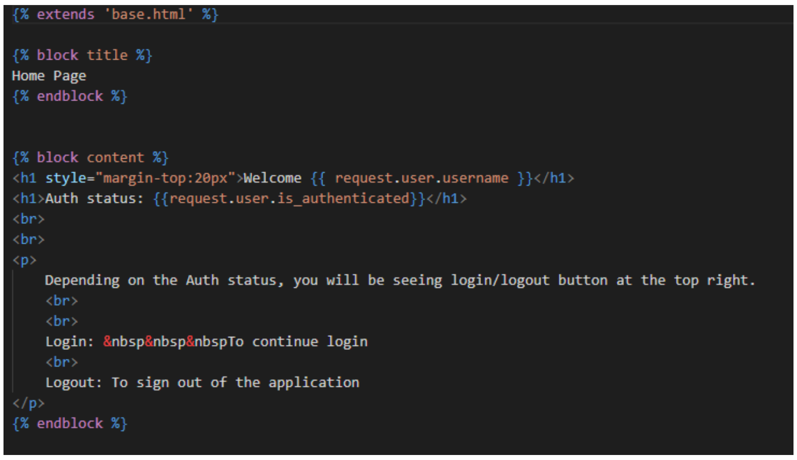


```html
<!doctype html>
<html lang="en">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
        integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

    {% load static %}

    <style type="text/css">
        @import url(https://fonts.googleapis.com/css?family=McLaren&display=swap);

        body {
            font-family: 'McLaren';
            font-style: normal;
            font-weight: 200;
            margin: 0px;
            background: #rgb(255,255,255);
        }
    </style>

    <title>
        {% block title %}

        {% endblock %}
    </title>
</head>

<body>

    <nav class="navbar navbar-expand navbar-dark bg-dark sticky-top">
        <a class="navbar-brand" href="{% url 'home' %}">
            <img src="{% static "images/crypto.jpg" %}" width="30" height="30" class="d-inline-block align-top"
```



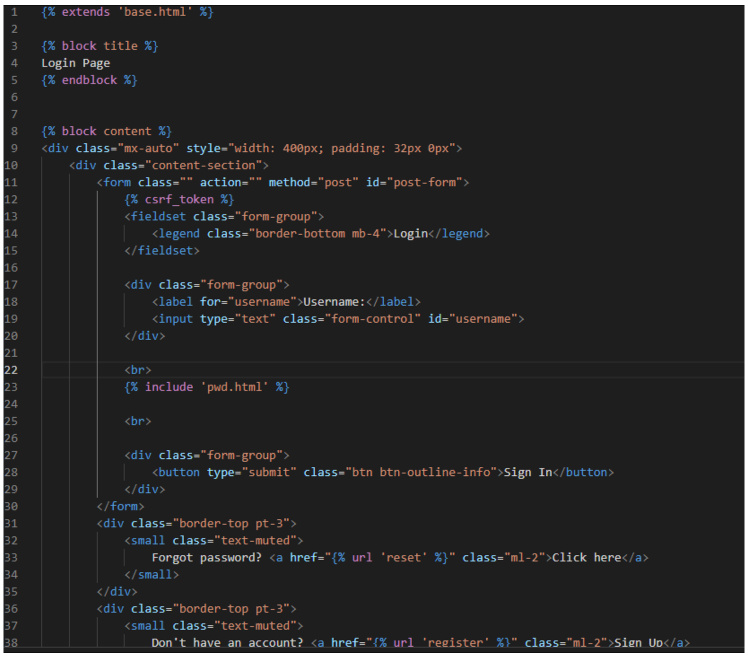
```
        </a>
        <div class="navbar-nav ml-auto">
            <a class="nav-item nav-link" href="{% url 'home' %}">Home</a>
            <a class="nav-item nav-link" href="{% url 'register' %}">Register</a>

            {% if request.user.is_authenticated %}
            <a class="nav-item nav-link" href="{% url 'logout' %}">Logout</a>
            {% else %}
            <a class="nav-item nav-link" href="{% url 'login' %}">Login</a>
            {% endif %}
        </div>
    </nav>

    <div class="container">
        {% if messages %}
        {% for message in messages %}
        <div class="alert alert-{{ message.tags }}">
            {{ message }}
        </div>
        {% endfor %}
        {% endif %}

        {% block content %}

        {% endblock %}
    </div>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
        integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
        integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous">
    </script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
        integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous">
```

11.home.html

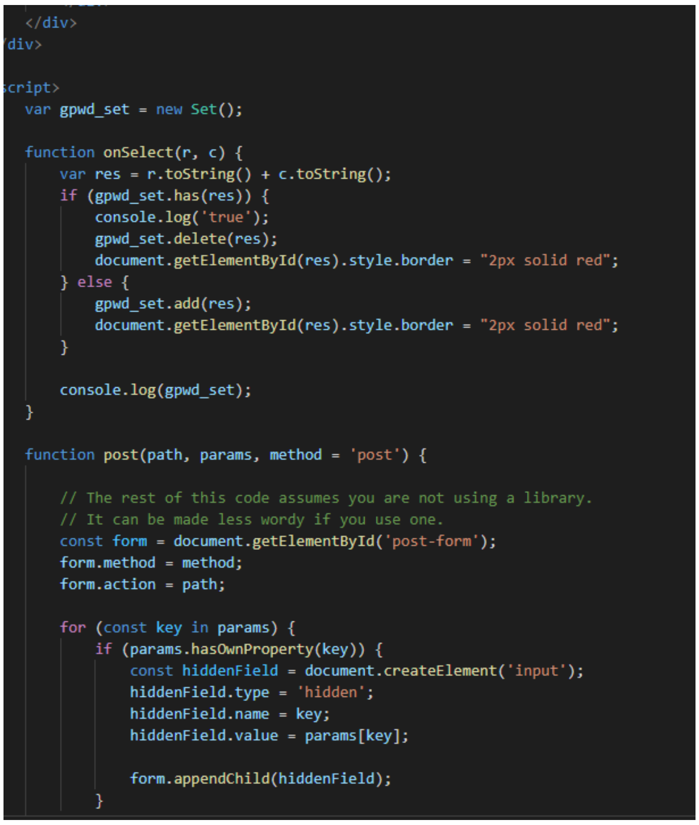


```
{% extends 'base.html' %}

{% block title %}
Home Page
{% endblock %}


{% block content %}
<h1 style="margin-top:20px">Welcome {{ request.user.username }}</h1>
<h1>Auth status: {{request.user.is_authenticated}}</h1>
<br>
<br>
<p>
    Depending on the Auth status, you will be seeing login/logout button at the top right.
    <br>
    <br>
    Login: &nbsp&nbsp&nbspTo continue login
    <br>
    Logout: To sign out of the application
</p>
{% endblock %}
```

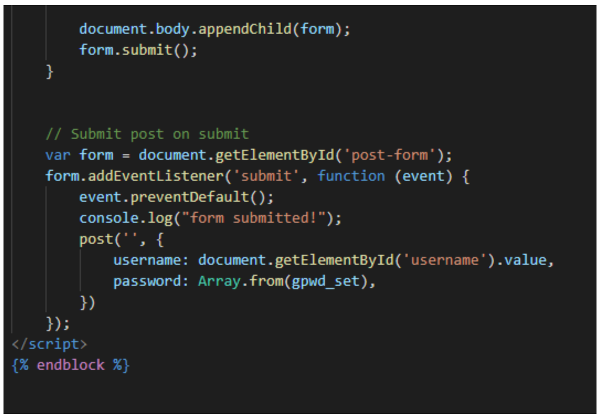
12.login.html



```
{% extends 'base.html' %}

{% block title %}
Login Page
{% endblock %}


{% block content %}
<div class="mx-auto" style="width: 400px; padding: 32px 0px">
    <div class="content-section">
        <form class="" action="" method="post" id="post-form">
            {% csrf_token %}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Login</legend>
            </fieldset>

            <div class="form-group">
                <label for="username">Username:</label>
                <input type="text" class="form-control" id="username">
            </div>

            <br>
            {% include 'pwd.html' %}

            <br>

            <div class="form-group">
                <button type="submit" class="btn btn-outline-info">Sign In</button>
            </div>
        </form>
        <div class="border-top pt-3">
            <small class="text-muted">
                Forgot password? <a href="{% url 'reset' %}" class="ml-2">Click here</a>
            </small>
        </div>
        <div class="border-top pt-3">
            <small class="text-muted">
                Don't have an account? <a href="{% url 'register' %}" class="ml-2">Sign Up</a>
```

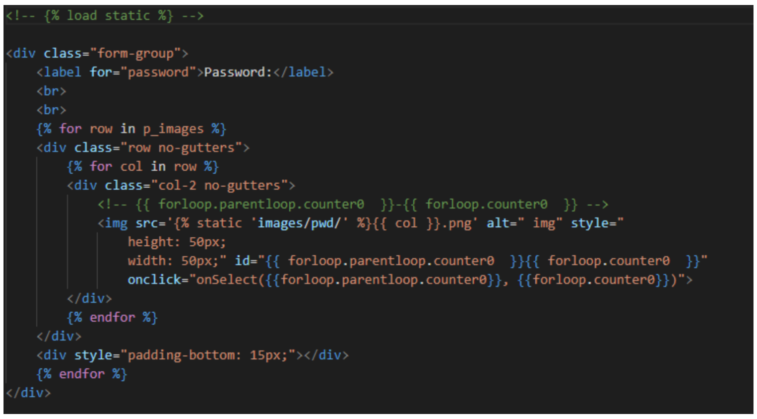


```
  </div>
/div>

script>
  var gpwd_set = new Set();

  function onSelect(r, c) {
      var res = r.toString() + c.toString();
      if (gpwd_set.has(res)) {
          console.log('true');
          gpwd_set.delete(res);
          document.getElementById(res).style.border = "2px solid red";
      } else {
          gpwd_set.add(res);
          document.getElementById(res).style.border = "2px solid red";
      }

      console.log(gpwd_set);
  }

  function post(path, params, method = 'post') {

      // The rest of this code assumes you are not using a library.
      // It can be made less wordy if you use one.
      const form = document.getElementById('post-form');
      form.method = method;
      form.action = path;

      for (const key in params) {
          if (params.hasOwnProperty(key)) {
              const hiddenField = document.createElement('input');
              hiddenField.type = 'hidden';
              hiddenField.name = key;
              hiddenField.value = params[key];

              form.appendChild(hiddenField);
          }
```



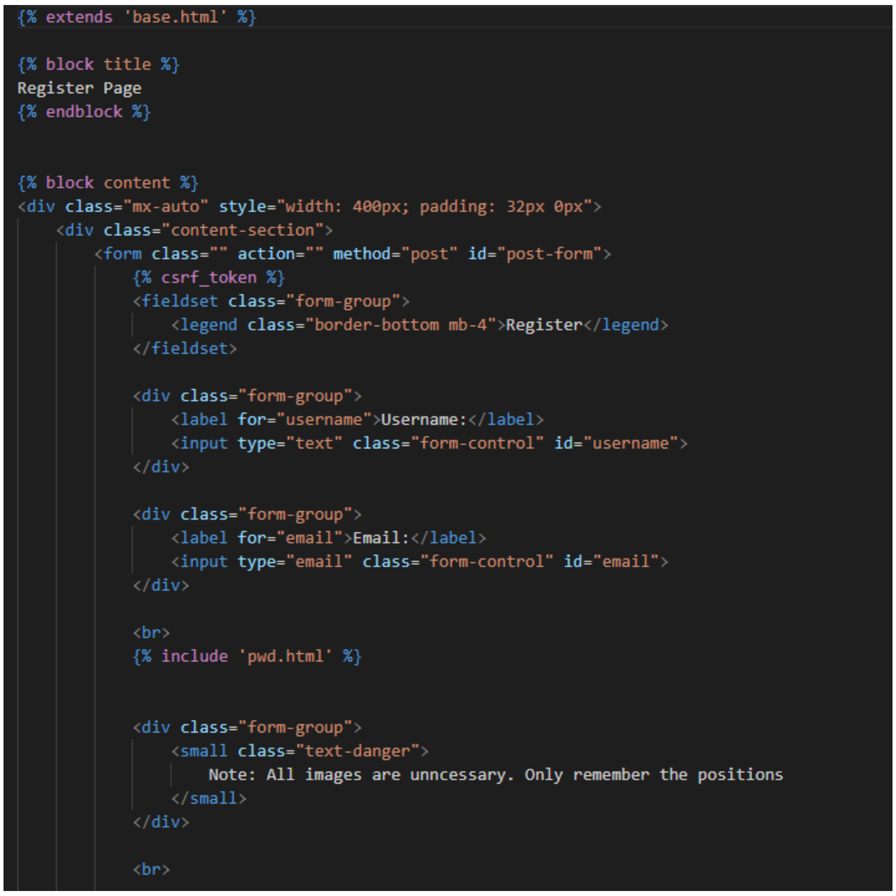
```
        document.body.appendChild(form);
        form.submit();
    }


    // Submit post on submit
    var form = document.getElementById('post-form');
    form.addEventListener('submit', function (event) {
        event.preventDefault();
        console.log("form submitted!");
        post('', {
            username: document.getElementById('username').value,
            password: Array.from(gpwd_set),
        })
    });
</script>
{% endblock %}
```

13.pwd.html

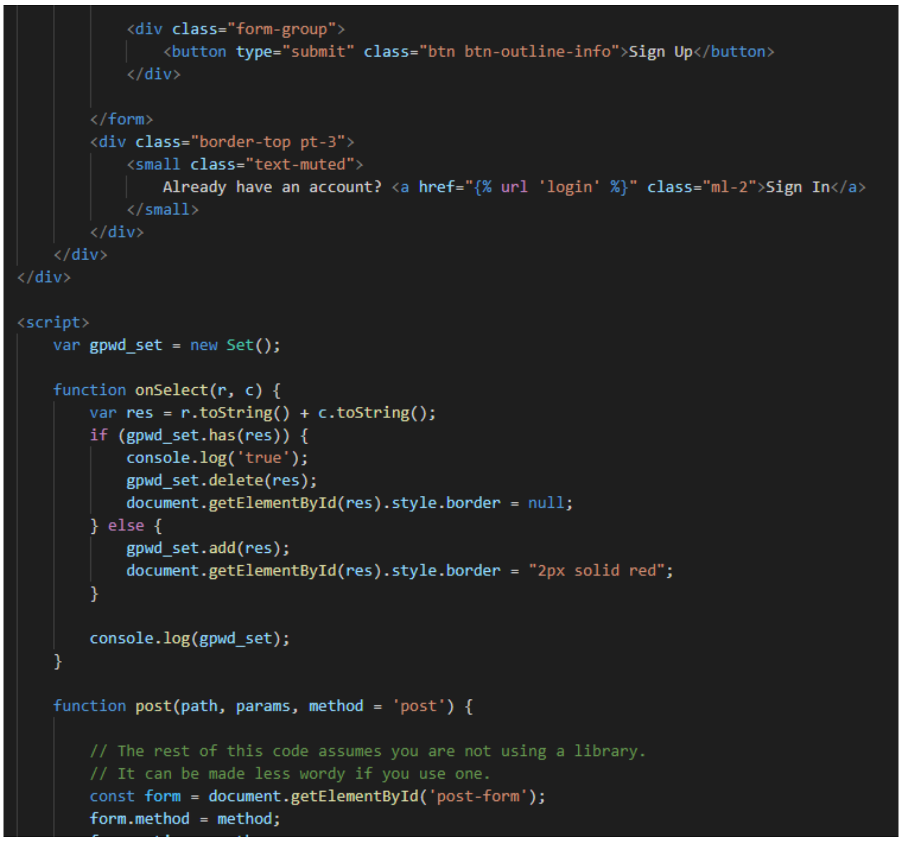


```
<!-- {% load static %} -->

<div class="form-group">
    <label for="password">Password:</label>
    <br>
    <br>
    {% for row in p_images %}
    <div class="row no-gutters">
        {% for col in row %}
        <div class="col-2 no-gutters">
            <!-- {{ forloop.parentloop.counter0  }}-{{ forloop.counter0  }} -->
            <img src='{% static 'images/pwd/' %}{{ col }}.png' alt=" img" style="
                height: 50px;
                width: 50px;" id="{{ forloop.parentloop.counter0  }}{{ forloop.counter0  }}"
                onclick="onSelect({{forloop.parentloop.counter0}}, {{forloop.counter0}})">
        </div>
        {% endfor %}
    </div>
    <div style="padding-bottom: 15px;"></div>
    {% endfor %}
</div>
```

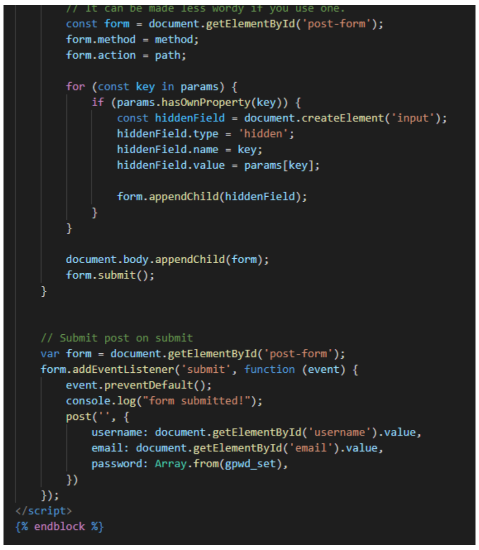
14.register.html



```
{% extends 'base.html' %}

{% block title %}
Register Page
{% endblock %}


{% block content %}
<div class="mx-auto" style="width: 400px; padding: 32px 0px">
    <div class="content-section">
        <form class="" action="" method="post" id="post-form">
            {% csrf_token %}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Register</legend>
            </fieldset>

            <div class="form-group">
                <label for="username">Username:</label>
                <input type="text" class="form-control" id="username">
            </div>

            <div class="form-group">
                <label for="email">Email:</label>
                <input type="email" class="form-control" id="email">
            </div>

            <br>
            {% include 'pwd.html' %}


            <div class="form-group">
                <small class="text-danger">
                    Note: All images are unncessary. Only remember the positions
                </small>
            </div>

            <br>
```

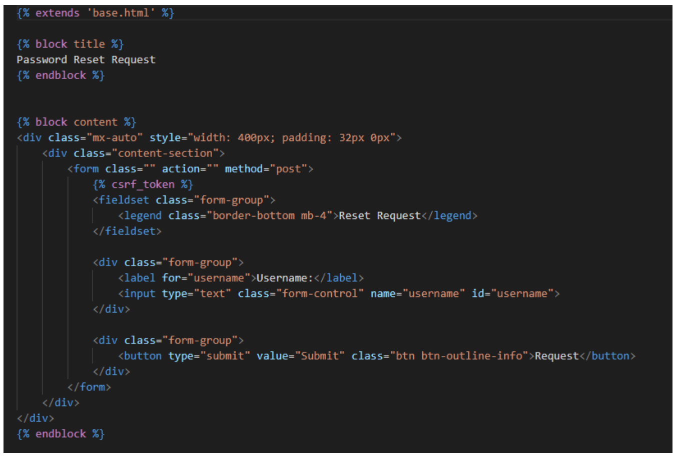


```
            <div class="form-group">
                <button type="submit" class="btn btn-outline-info">Sign Up</button>
            </div>

        </form>
        <div class="border-top pt-3">
            <small class="text-muted">
                Already have an account? <a href="{% url 'login' %}" class="ml-2">Sign In</a>
            </small>
        </div>
    </div>
</div>

<script>
    var gpwd_set = new Set();

    function onSelect(r, c) {
        var res = r.toString() + c.toString();
        if (gpwd_set.has(res)) {
            console.log('true');
            gpwd_set.delete(res);
            document.getElementById(res).style.border = null;
        } else {
            gpwd_set.add(res);
            document.getElementById(res).style.border = "2px solid red";
        }

        console.log(gpwd_set);
    }

    function post(path, params, method = 'post') {

        // The rest of this code assumes you are not using a library.
        // It can be made less wordy if you use one.
        const form = document.getElementById('post-form');
        form.method = method;
```



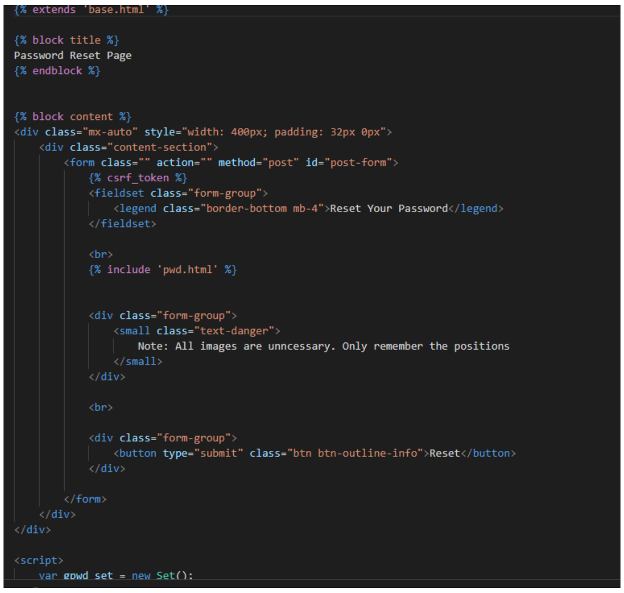
```
        // It can be made less wordy if you use one.
        const form = document.getElementById('post-form');
        form.method = method;
        form.action = path;

        for (const key in params) {
            if (params.hasOwnProperty(key)) {
                const hiddenField = document.createElement('input');
                hiddenField.type = 'hidden';
                hiddenField.name = key;
                hiddenField.value = params[key];

                form.appendChild(hiddenField);
            }
        }

        document.body.appendChild(form);
        form.submit();
    }


    // Submit post on submit
    var form = document.getElementById('post-form');
    form.addEventListener('submit', function (event) {
        event.preventDefault();
        console.log("form submitted!");
        post('', {
            username: document.getElementById('username').value,
            email: document.getElementById('email').value,
            password: Array.from(gpwd_set),
        })
    });
</script>
{% endblock %}
```

15.reset_request.html

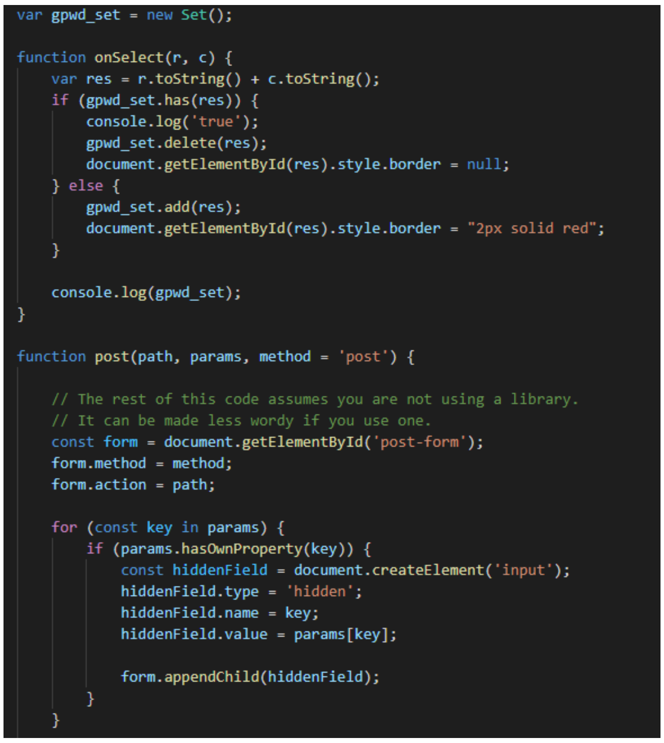


```
{% extends 'base.html' %}

{% block title %}
Password Reset Request
{% endblock %}


{% block content %}
<div class="mx-auto" style="width: 400px; padding: 32px 0px">
    <div class="content-section">
        <form class="" action="" method="post">
            {% csrf_token %}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Reset Request</legend>
            </fieldset>

            <div class="form-group">
                <label for="username">Username:</label>
                <input type="text" class="form-control" name="username" id="username">
            </div>

            <div class="form-group">
                <button type="submit" value="Submit" class="btn btn-outline-info">Request</button>
            </div>
        </form>
    </div>
</div>
{% endblock %}
```

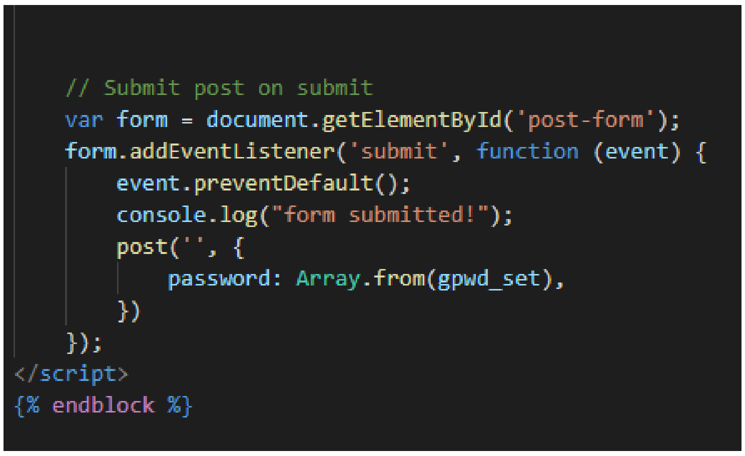
16.reset.html



```
{% extends 'base.html' %}

{% block title %}
Password Reset Page
{% endblock %}


{% block content %}
<div class="mx-auto" style="width: 400px; padding: 32px 0px">
    <div class="content-section">
        <form class="" action="" method="post" id="post-form">
            {% csrf_token %}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Reset Your Password</legend>
            </fieldset>

            <br>
            {% include 'pwd.html' %}


            <div class="form-group">
                <small class="text-danger">
                    Note: All images are unncessary. Only remember the positions
                </small>
            </div>

            <br>

            <div class="form-group">
                <button type="submit" class="btn btn-outline-info">Reset</button>
            </div>

        </form>
    </div>
</div>

<script>
    var gpwd_set = new Set();
```

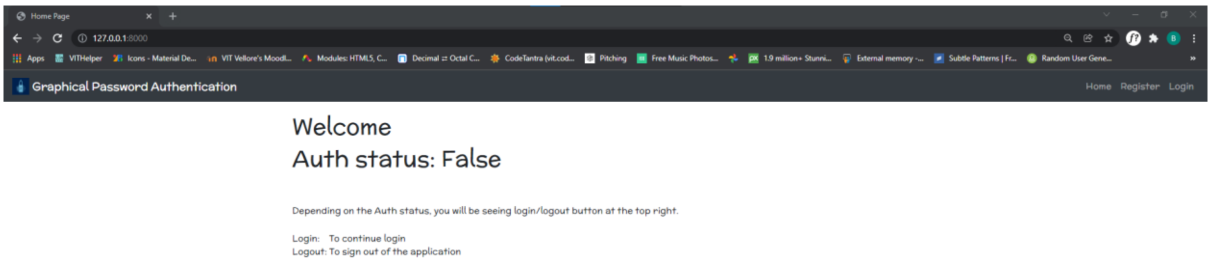


```
var gpwd_set = new Set();

function onSelect(r, c) {
    var res = r.toString() + c.toString();
    if (gpwd_set.has(res)) {
        console.log('true');
        gpwd_set.delete(res);
        document.getElementById(res).style.border = null;
    } else {
        gpwd_set.add(res);
        document.getElementById(res).style.border = "2px solid red";
    }

    console.log(gpwd_set);
}

function post(path, params, method = 'post') {

    // The rest of this code assumes you are not using a library.
    // It can be made less wordy if you use one.
    const form = document.getElementById('post-form');
    form.method = method;
    form.action = path;

    for (const key in params) {
        if (params.hasOwnProperty(key)) {
            const hiddenField = document.createElement('input');
            hiddenField.type = 'hidden';
            hiddenField.name = key;
            hiddenField.value = params[key];

            form.appendChild(hiddenField);
        }
    }
```



```


    // Submit post on submit
    var form = document.getElementById('post-form');
    form.addEventListener('submit', function (event) {
        event.preventDefault();
        console.log("form submitted!");
        post('', {
            password: Array.from(gpwd_set),
        })
    });
</script>
{% endblock %}
```

OUTPUT SS



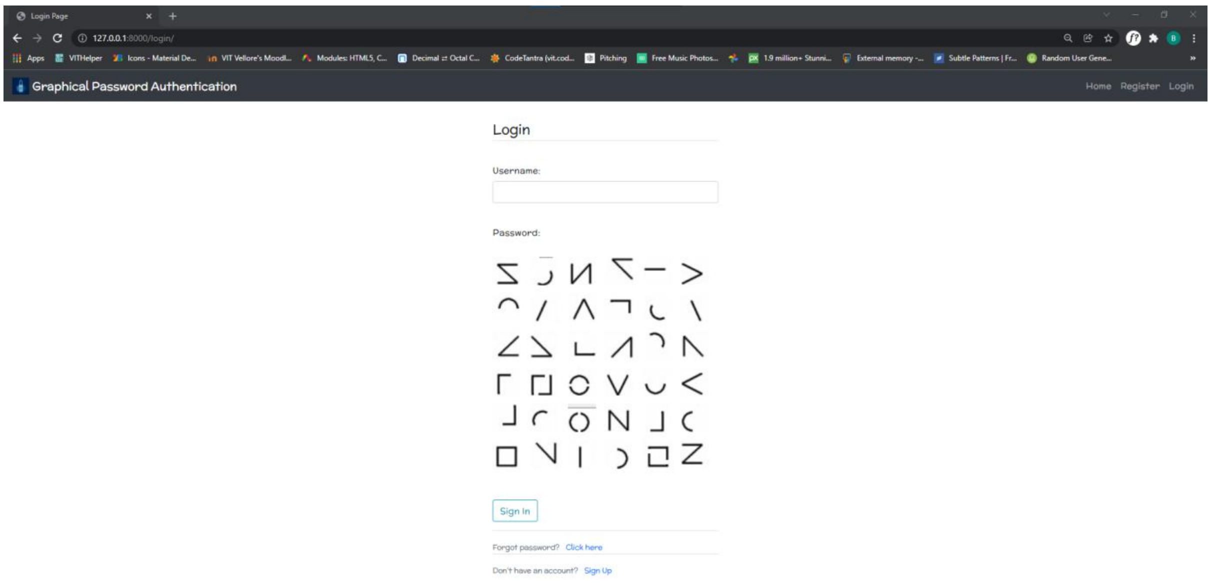
Graphical Password Authentication

Home Register Login

Welcome
Auth status: False

Depending on the Auth status, you will be seeing login/logout button at the top right.

Login: To continue login
Logout: To sign out of the application



Graphical Password Authentication

Home Register Login

Login

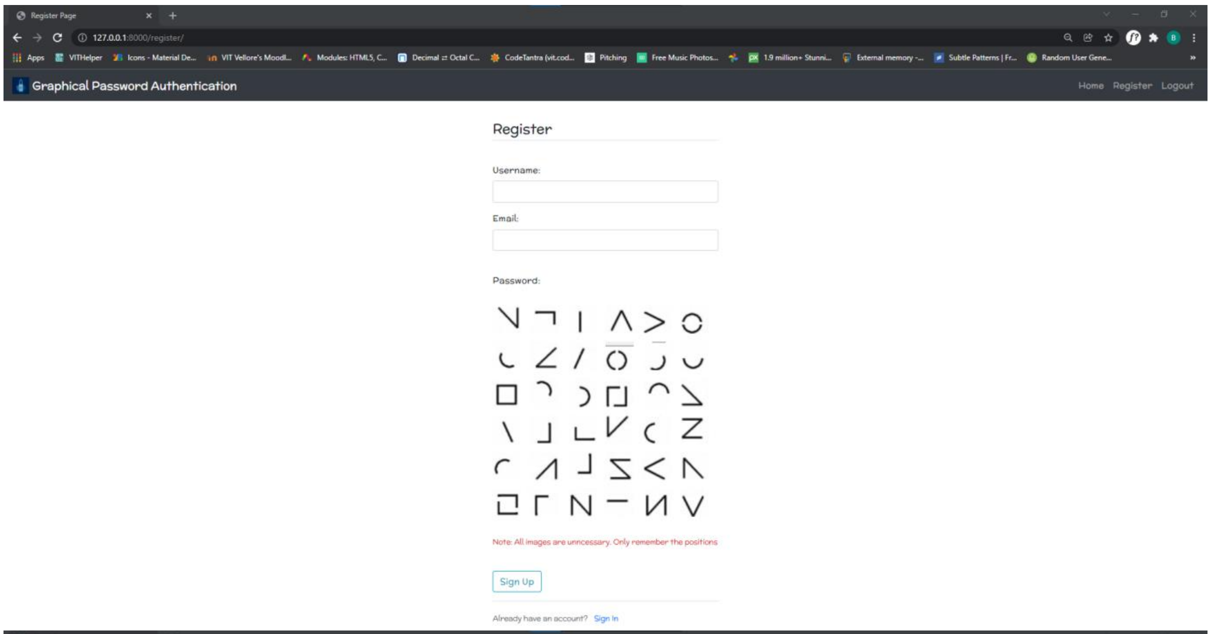
Username:

Password:

Sign In

Forgot password? Click here

Don't have an account? Sign Up



Graphical Password Authentication

Home Register Logout

Register

Username:

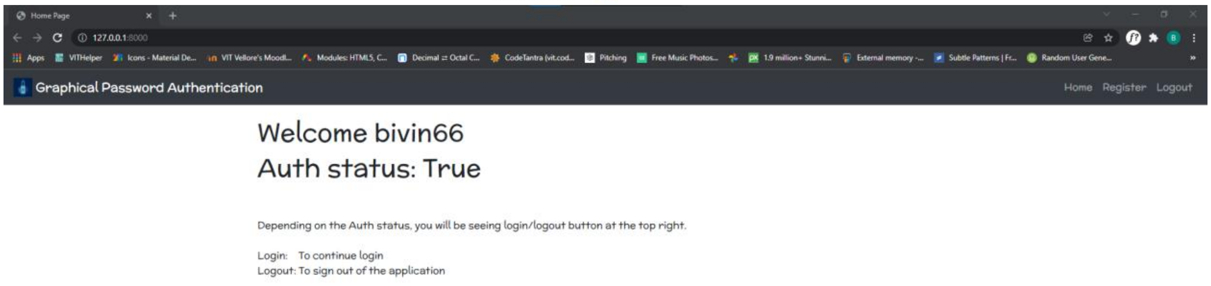
Email:

Password:

Note: All images are unncessary. Only remember the positions

Sign Up

Already have an account? Sign In



Graphical Password Authentication

Home Register Logout

Welcome bivin66
Auth status: True

Depending on the Auth status, you will be seeing login/logout button at the top right.

Login: To continue login
Logout: To sign out of the application