

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ
ЕНЕРГЕТИКИ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В
ЕНЕРГЕТИЦІ

КУРСОВА РОБОТА

з дисципліни: ««Основи Веб-програмування»

на тему: «Додаток, який створює сильні та безпечні паролі»

Керівник:

Гагарін О. О.

«Допущено до захисту»

(особистий підпис керівника)

«__» _____ 2025 р.

Захищено з оцінкою

(оцінка)

Виконав:

Домарацький Дмитро Олександрович
студент 2 курсу

групи ТВ-33

(особистий підпис виконавця)

«__» _____ 2025 р.

Посилання на **GitHub** репозиторій <https://github.com/dmtrDO/WebCourse>

Київ — 2025

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Напрямок підготовки 121 Інженерія програмно забезпечення

ЗАВДАННЯ
НА КУРСОВУ РОБОТУ СТУДЕНТУ
Домарацькому Дмитру Олександровичу

1. Тема роботи — «Додаток, який створює сильні та безпечні паролі»
Керівник курсової роботи — Недашківський О.Л.
2. Строк подання студентом роботи: «25» травня 2025 р.
3. Вихідні дані до проекту (роботи): мова — Python, дизайн — html, css, js;
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): розробити додаток для генерації паролів мовою Python;
5. Дата видачі завдання: 4 лютого 2025.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання курсової роботи	Строк виконання етапів проекту
1.	Отримання і затвердження теми курсової роботи	14.01.2025
2.	Створення UML діаграм веб додатка	01.02.2025
3.	Проектування та реалізація моделей даних	12.02.2025
4.	Розміщення і стилізація елементів інтерфейсу	27.02.2025
5.	Розробка БД для збереження та отримання інформації	05.03.2025
6.	Розробка механізмів отримання та оновлення даних	14.03.2025
7.	Розробка БД для збереження та отримання інформації	28.03.2025
8.	Інтеграція сторонніх сервісів в систему	06.04.2025
9.	Створення серверної архітектури додатку	20.04.2025
10.	Реалізація обробки запитів і взаємодії з БД на сервері	01.05.2025
11.	Тестування та налаштування веб додатку на сервері	13.05.2025
12.	Оформлення записки до курсової роботи	22.05.2025

Студент _____ Домарацький Д.О.
(підпис)

Керівник курсової роботи _____ Недашківський О.Л.
(підпис)

АНОТАЦІЯ

Під час виконання цієї курсової роботи було освоєно основи вебпрограмування, які були використані для створення вебзастосунку, що дозволяє генерувати надійні паролі. Метою створення такого застосунку є допомога користувачам у формуванні унікальних і складних паролів, які важко зламати. Згенеровані паролі можна використовувати, зокрема, під час реєстрації на різних вебсайтах або в додатках, що забезпечує захист персональних даних.

Основним завданням було спроектувати застосунок, продумавши як логічну частину (бекенд), так і візуальну (фронтенд) з подальшою реалізацією. У якості основного інструменту розробки було обрано Python-фреймворк Django, який надає низку вбудованих функцій, а саме: адміністративну частину сайту, ORM (взаємодія з базою даних через Python-код), систему шаблонів HTML-сторінок, URL-навігацію, роботу з формами, кешування сесій (через `request.session`) тощо. Середовищем розробки став PyCharm Community 2024.3.1.1.

У межах цієї роботи реалізовано адаптивний, кросплатформений вебзастосунок, який коректно працює на пристроях з різними розмірами екранів — від мобільних телефонів до великих моніторів. Дизайн мінімалістичний і побудований на поєднанні трьох основних кольорів, із зручною навігацією. Основною метою є швидкість використання: користувач може одразу отримати пароль та скопіювати його. Для розширених можливостей — таких як збереження історії паролів — передбачена функція реєстрації.

ANNOTATION

During the implementation of this course project, the basics of web development were mastered and applied to create a web application that generates strong passwords. The purpose of this application is to help users create unique and complex passwords that are difficult to crack. The generated passwords can be used, for example, during registration on various websites or in applications, thereby ensuring the protection of personal data.

The main task was to design the application by developing both the logical part (backend) and the visual part (frontend), followed by their implementation. The main development tool chosen was the Django Python framework, which provides a number of built-in features, such as an administrative interface, ORM (database interaction through Python code), HTML template system, URL routing, form handling, session caching (via `request.session`), and more. The development environment was PyCharm Community 2024.3.1.1.

As part of this work, a responsive, cross-platform web application was implemented, functioning correctly on devices with different screen sizes — from mobile phones to large monitors. The design is minimalist and based on a combination of three main colors, with intuitive navigation. The primary goal is ease of use: the user can immediately generate a password and copy it. For advanced functionality — such as saving password history — a registration feature is available.

ЗМІСТ

АНОТАЦІЯ.....	3
ANNOTATION.....	4
ЗМІСТ.....	5
ВСТУП.....	6
1 АНАЛІЗ ТА ПРОЄКТУВАННЯ.....	7
1.1 Створення діаграм компонентів, взаємодії та класів вебдодатку.....	7
1.2 Проектування структури меню з урахуванням потреб користувача.....	10
1.3 Розміщення та стилізація елементів інтерфейсу.....	11
1.4 Проектування та реалізація моделей даних для збереження інформації...12	
2 РОЗРОБКА ТА РЕАЛІЗАЦІЯ.....	14
2.1 Розробка механізмів отримання та оновлення даних.....	14
2.2 Розробка бази даних для збереження та отримання інформації.....	15
2.3 Інтеграція сторонніх сервісів в систему.....	17
2.4 Створення серверної архітектури додатку.....	20
2.5 Реалізація обробки запитів та взаємодії з базою даних на сервері.....	22
2.6 Тестування та налаштування веб додатку на сервері.....	23
2.7 Функція генерації.....	26
3 ІНСТРУКЦІЯ КОРИСТУВАЧА.....	29
ВИСНОВКИ.....	37
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	38
ДОДАТОК.....	39

ВСТУП

У сучасному цифровому середовищі безпека персональних даних стає все більш актуальною проблемою. Щодня користувачі створюють нові акаунти на різноманітних платформах, і саме слабкі паролі часто стають причиною витоку даних або несанкціонованого доступу до облікових записів. Саме тому питання формування складних, унікальних та надійних паролів набуває особливої ваги.

Актуальність теми даної курсової роботи полягає в необхідності створення інструменту, який би допомагав користувачам генерувати захищені паролі без додаткових знань у сфері інформаційної безпеки. Розробка вебзастосунку з такою функціональністю дозволяє зробити процес створення паролів швидким, зручним і безпечним.

Метою роботи є створення кросплатформеного вебзастосунку для генерації надійних паролів із можливістю їх копіювання та збереження, а також забезпечення адаптивного інтерфейсу для комфортного користування на різних пристроях.

Об'єктом дослідження є процес створення веборієнтованого програмного забезпечення, предметом — алгоритми генерації паролів та реалізація зручного користувацького інтерфейсу.

У роботі застосовано такі методи: структурний аналіз вимог, проєктування архітектури вебзастосунку, реалізація бекенд- і фронтенд-частини за допомогою технологій Django, HTML, CSS та JavaScript.

Курсова робота складається з трьох основних розділів: аналіз та проєктування, розробка та реалізація, інструкція користувача.

1 АНАЛІЗ ТА ПРОЄКТУВАННЯ

1.1 Створення діаграм компонентів, взаємодії та класів вебдодатку

Для початку проаналізуємо основну задачу. Для цього окреслимо функціонал майбутнього веб-додатку і створимо візуальні компоненти. Будемо використовувати UML — мову широкого профілю, яка використовує графічні позначення для створення абстрактної моделі системи. Почнемо з діаграми прецедентів (рисунок 1.1.1), яка показує різні варіанти використання та різні типи користувачів системи.

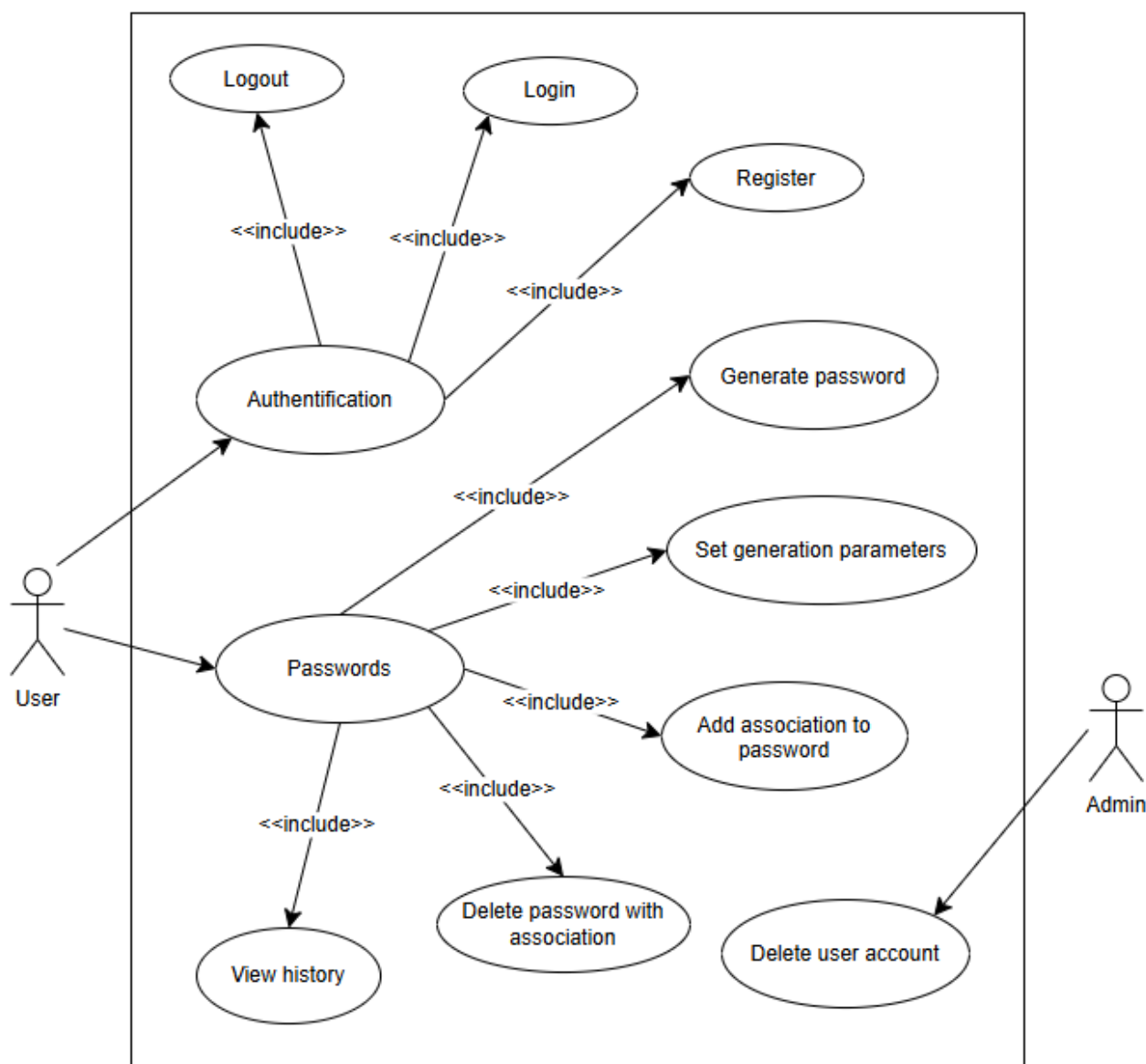


Рисунок 1.1.1 — Діаграма варіантів використання

Веб-додаток дає змогу користувачам створювати обліковий запис та входити в нього, генерувати безпечні паролі з можливістю їхньої кастомізації (довжина, символи, регістр, спецсимволи тощо), переглядати згенеровані паролі та надавати їм асоціації, також він може видалити якийсь пароль зі списку. Адміністратор має змогу видаляти облікові записи у разі потреби. Система забезпечує зручний інтерфейс користувача а також зберігає інформацію про акаунт із застосуванням заходів безпеки через хешування і шифрування паролів.

Далі створимо діаграму класів (рисунок 1.1.2), з допомогою якої можна роздивитись статичні елементи системи, їх зміст та відношення між ними.

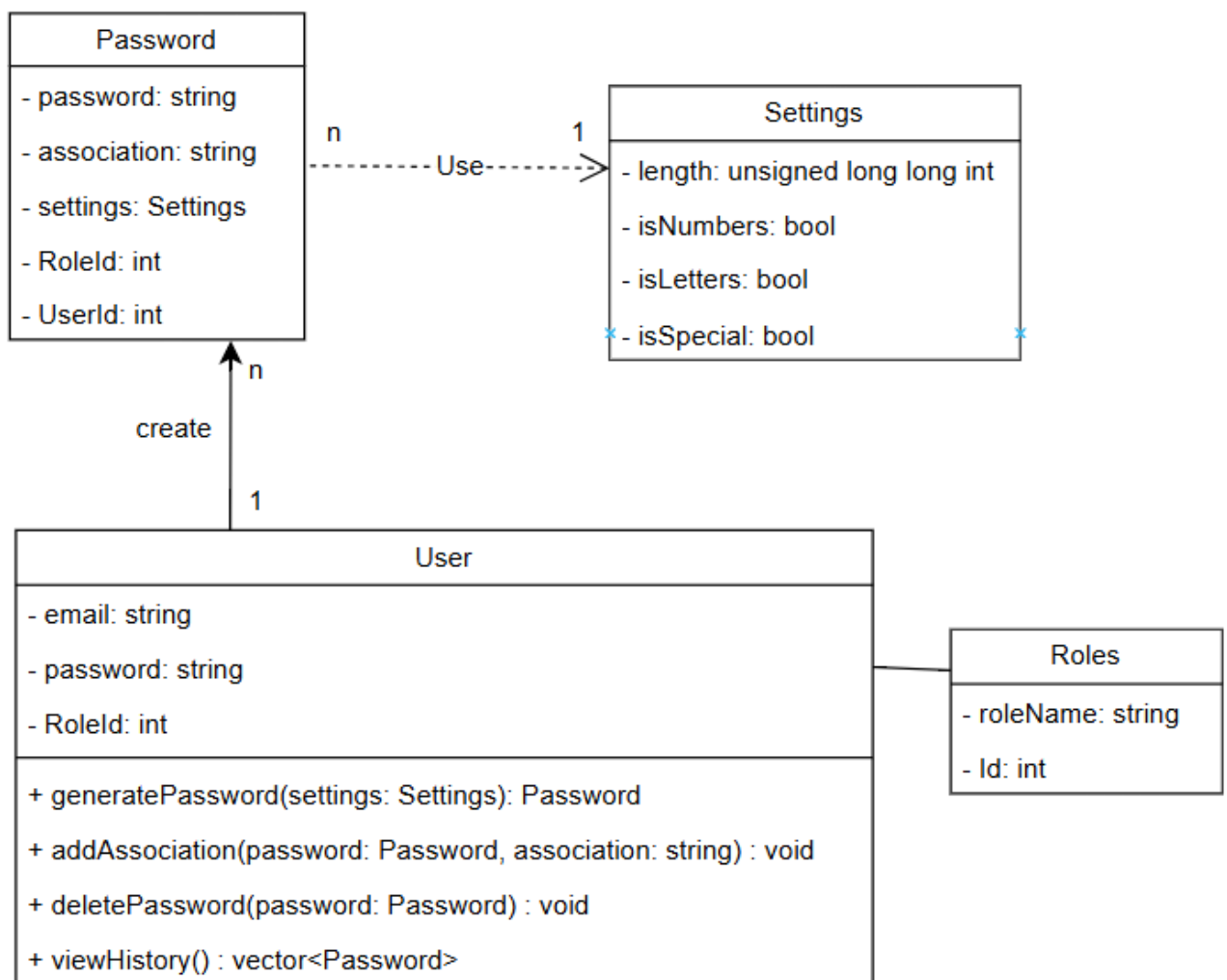


Рисунок 1.1.2 — Діаграма класів

Тепер нам потрібно відобразити компоненти системи і залежності між ними, включаючи зовнішні та внутрішні компоненти, наприклад, база даних, браузер тощо. Для цього створимо діаграму компонентів (рисунок 1.1.3).

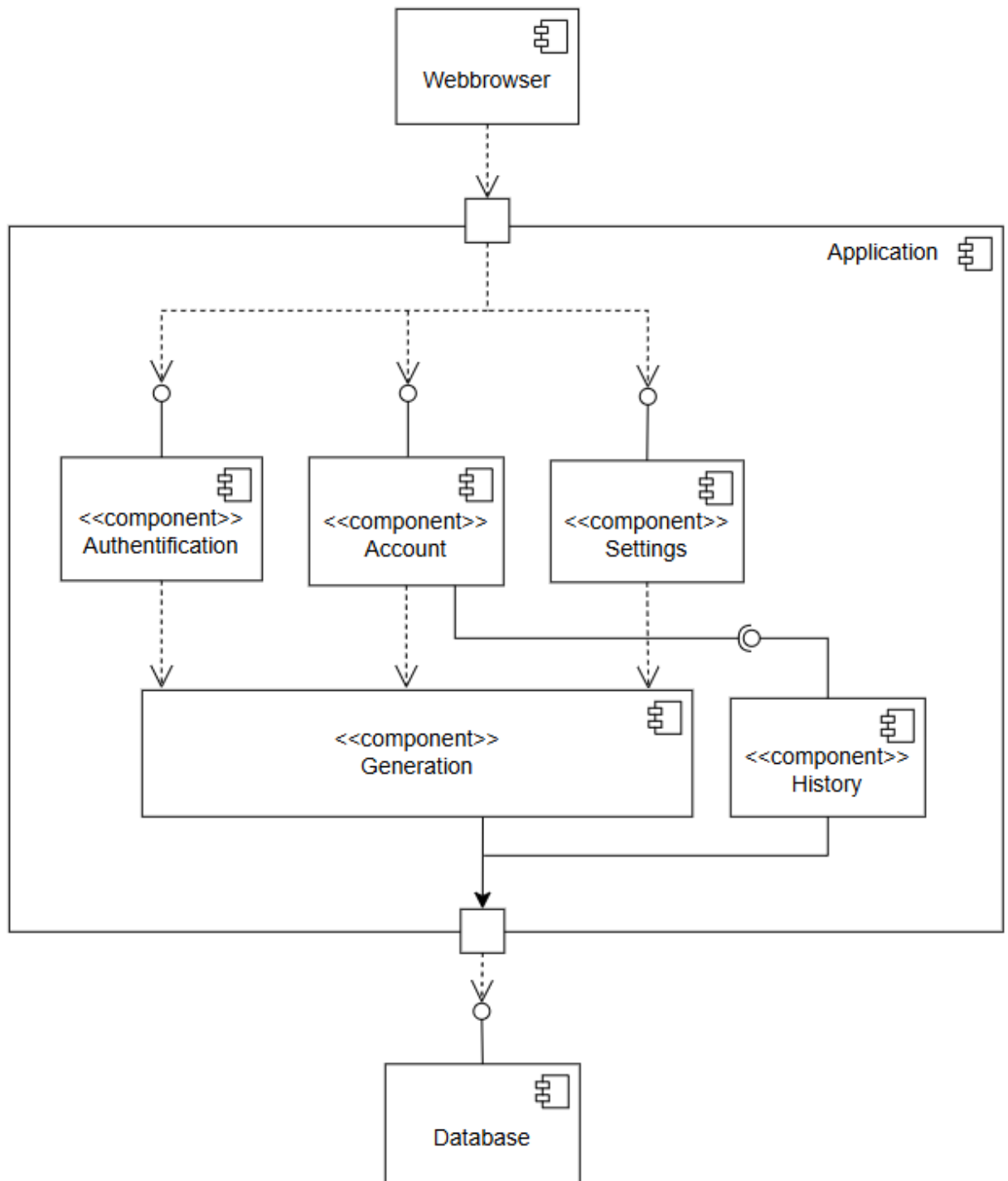


Рисунок 1.1.3 — Діаграма компонентів

1.2 Проектування структури меню з урахуванням потреб користувача

Меню додатку для створення паролів буде включати наступні пункти:

1. Створити пароль;
2. Історія паролів;
3. Параметри генерації;
4. Обліковий запис;
5. Головна.

Спроекуємо каркас головної сторінки для незареєстрованого та зареєстрованого користувача (рисунок 1.2.1).

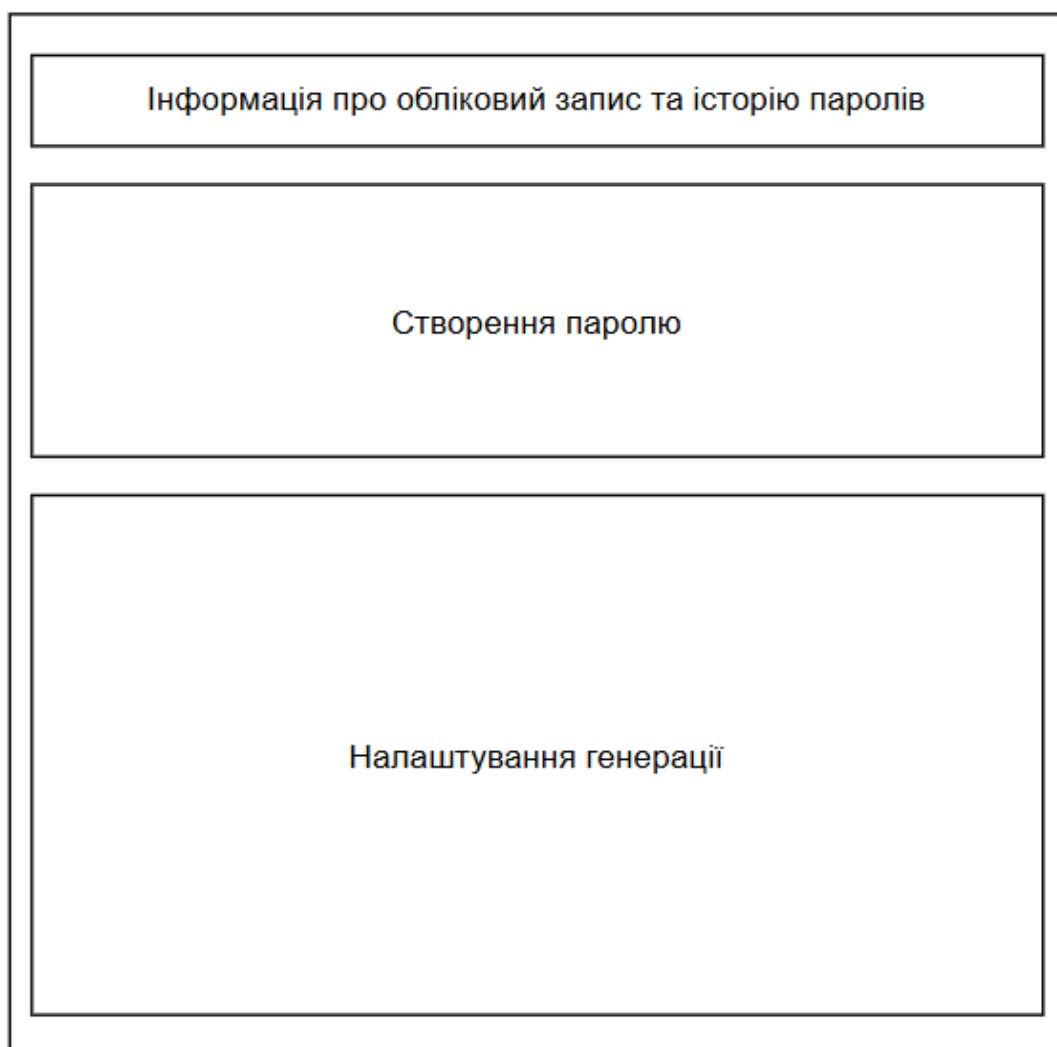
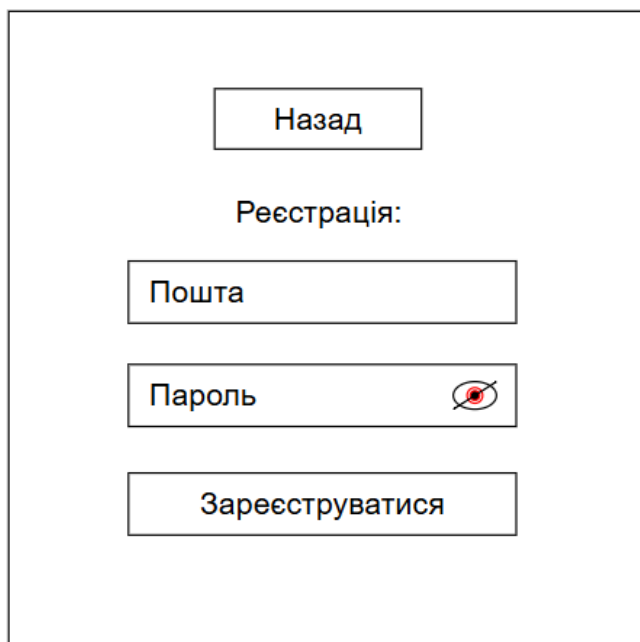


Рисунок 1.2.1 — Схематичне зображення головної сторінки у веб-додатку

1.3 Розміщення та стилізація елементів інтерфейсу


Сторінка створення акаунту користувача буде містити поля для вводу пошти та паролю, та дві кнопки для повернення на головну сторінку (рисунок 1.3.1).



Назад

Реєстрація:

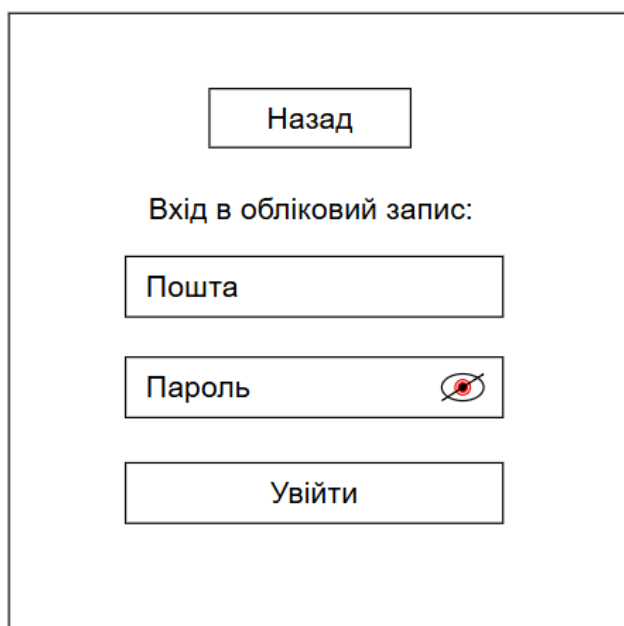
Пошта

Пароль 

Зареєструватися

Рисунок 1.3.1 — Схема розміщення елементів інтерфейсу сторінки реєстрації


Сторінка авторизації користувача буде містити аналогічні поля та кнопки, але з відповідними написами про вхід (рисунок 1.3.2).



Назад

Вхід в обліковий запис:

Пошта

Пароль 

Увійти

Рисунок 1.3.2 — Схема розміщення елементів інтерфейсу сторінки авторизації

1.4 Проектування та реалізація моделей даних для збереження інформації

В системі передбачено наступні сутності даних:

1. Користувачі;
2. Налаштування генерації;
3. Згенеровані паролі;
4. Ролі.

Схематично модель представлена на рисунку 1.4.1.

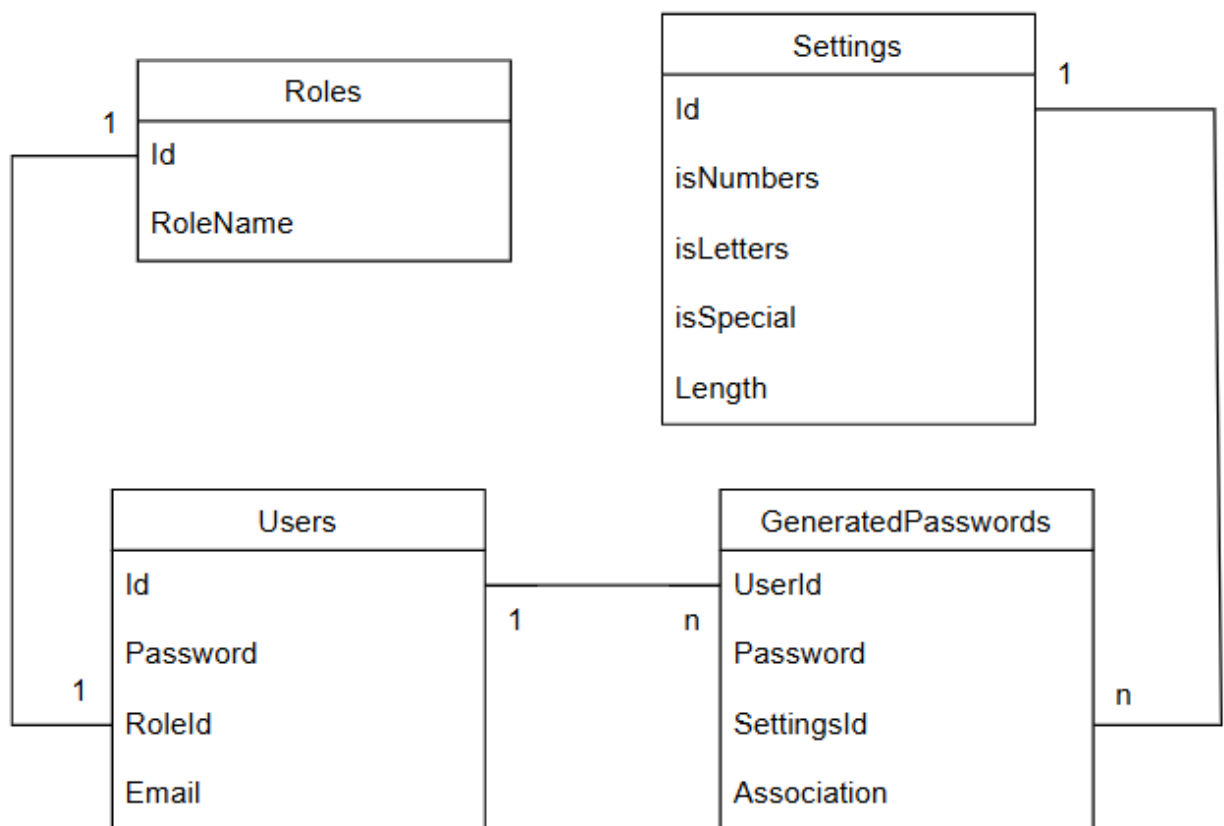


Рисунок 1.4.1 — Діаграма моделей даних для збереження інформації

Модель даних “Users” зберігає дані облікового запису: пошта та пароль, а “Settings” – дані генерації паролю: довжину, змінні булевого типу для цифр, літер та спеціальних символів. Також є таблиця згенерованих паролів, яка містить айді відповідних таблиць, згенерований пароль та асоціацію до паролю

(за замовчуванням пустий рядок ‘’). Таблиця ролі містить назву ролі користувача.

Залежності між моделями ‘User’ та ‘GeneratedPasswords’ один до багатьох, оскільки один користувач може мати багато паролів. Зв’язок між ‘GeneratedPasswords’ та ‘Settings’ – багато до одного, оскільки декілька паролів можуть мати однакові параметри генерації. Між ролями та користувачем зв’язок один до одного, оскільки один користувач може мати тільки одну роль.

Слід зазначити, що у таблиці Roles містяться ідентифікатори ролей (ID, назва ролі). У таблиці Users зберігаються як адміністратори, так і звичайні користувачі, але їхню роль визначають за допомогою foreign key з таблиці Roles. Як правило, на великих проєктах створюють окрему таблицю UsersRoles, де визначаються всі можливі ролі користувачів. Тобто на ентерпрайз проєктах зазвичай застосовують другий підхід (з таблицею UsersRoles), і він вважається більш коректним, проте в рамках даної роботи будемо мати тільки таблицю Roles.

2 РОЗРОБКА ТА РЕАЛІЗАЦІЯ

2.1 Розробка механізмів отримання та оновлення даних

Усі обробки запитів реалізовано у `views.py` (див. Додаток).

Реєстрація користувача. Після натискання кнопки "Зареєструватися", дані з форми надсилаються методом POST на сервер за шляхом `/registration`. У тілі запиту передаються дані користувача: електронна пошта, пароль. Django обробляє ці дані через відповідне `view`, перевіряє їх на коректність і створює нового користувача. У випадку успішної реєстрації користувач перенаправляється на сторінку входу. Якщо введені дані некоректні (наприклад, пароль занадто короткий), користувачу повертається та сама форма з повідомленням про помилку, причому помилки, пов'язані з БД, обробляються у `views.py`, а помилки, пов'язані з довжиною паролів, або невірним форматом електронної пошти, або невірними символами при заповненні паролю і т.д., то такі помилки оброблюються через `javascript`.

Авторизація користувача. Форма входу надсилає дані методом POST на шлях `/login`. У формі передаються пошта і пароль. Django перевіряє ці дані аналогічно, як при реєстрації, і, якщо вони правильні — виконує вхід користувача, після чого його перенаправляють на головну сторінку застосунку.

Генерація паролів. При натисканні на кнопку «Згенерувати» на сервер надсилається POST-запит тим же шляхом, з якого він надсилається, попередньо обробивши вхідні налаштування через сесії та передавши ці дані в БД.

Перегляд історії. Для отримання історії збережених паролів реалізовано `view`, який при GET-запиті до URL `/history` витягує всі записи паролів поточного користувача з бази даних та передає їх у шаблон для відображення. Таким чином реалізується механізм отримання даних.

При спробі додати опис до паролю, у шаблоні є тег поля вводу, значення якого задається через `js` (`id` та асоціація), а при відправці запиту до сервера, відповідний метод `view` обробляє його і повертає назад за тим же шляхом.

2.2 Розробка бази даних для збереження та отримання інформації

У розробленому вебзастосунку використовується реляційна база даних, побудована за допомогою ORM (Object-Relational Mapping), яку надає фреймворк Django. Для зберігання та обробки даних обрано СКБД SQLite, оскільки вона є вбудованою і не потребує окремої установки та забезпечує достатній рівень функціональності.

Схема (в більшості IDE чи програмах для роботи з базами даних схеми генеруються автоматично на основі вже створених таблиць) (рисунок 2.2.1).

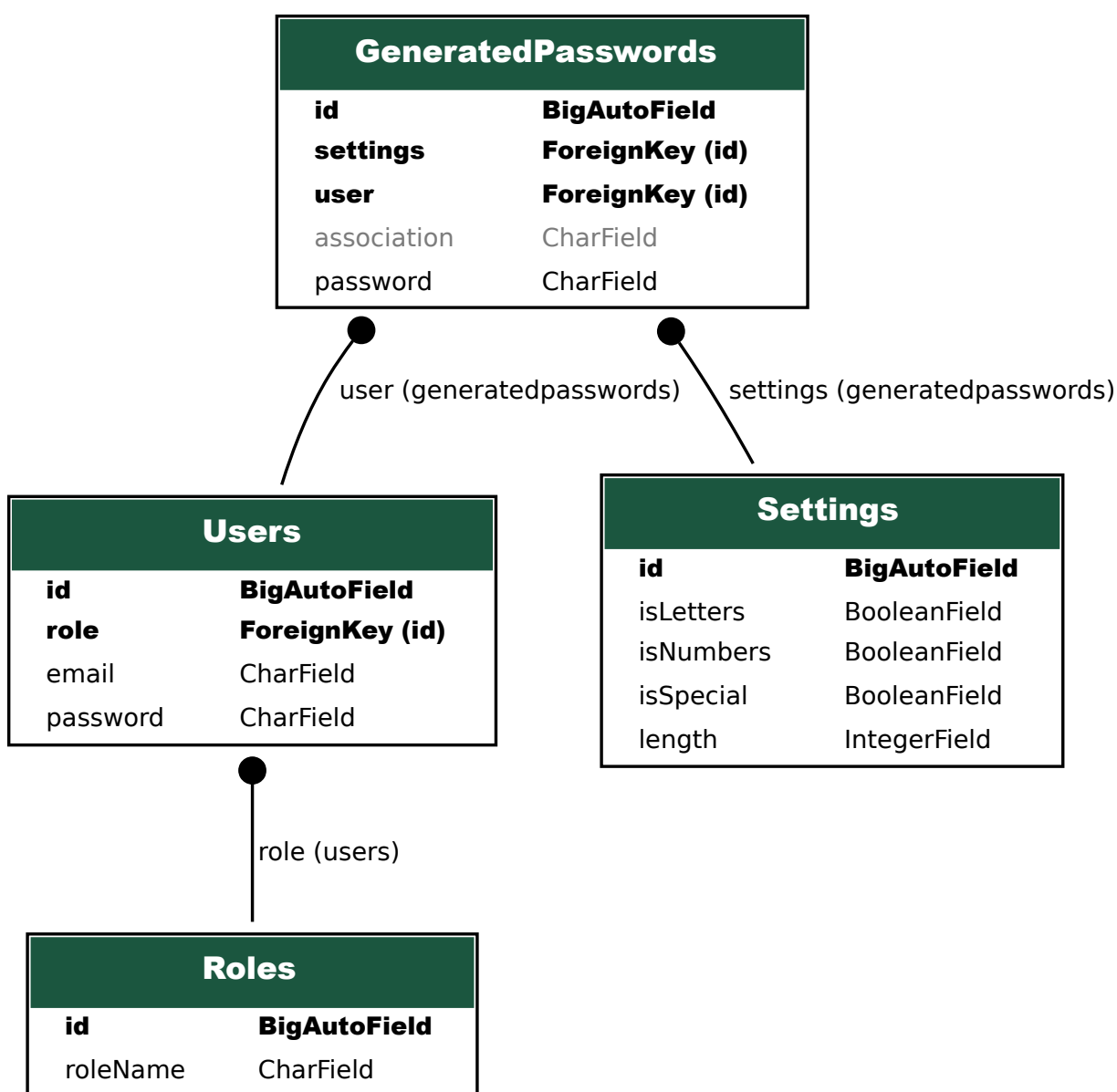


Рисунок 2.2.1 — Схема БД

Тут для генерації схеми бази даних було використано команду в терміналі PyCharm:

```
python manage.py graph_models main -o diagram.svg
```

Поле email у таблиці User має унікальний індекс, що забезпечує швидкий пошук і перевірку унікальності. Зовнішні ключі (user, settings, role) автоматично індексуються Django, що забезпечує ефективність при фільтрації.

Важливо зазначити, що значення поля password у БД зберігається не у явному вигляді. У базу даних, а саме у таблицю Users, потрапляє тільки хеш паролю, який створюється при надсиланні користувачем відповідного запиту для його реєстрації. І це дуже важливо, оскільки навіть адміністратори БД не можуть побачити паролі користувачів, що гарантує безпеку конфіденційних даних. Досягається це шляхом одностороннього хешування з використанням функцій make_password та check_password [1].

Поле password таблиці GeneratedPassword теж не зберігає пароль в тому вигляді, в якому він був згенерований користувачем. Він шифрується за допомогою криптографічного стандарту Fernet з бібліотеки cryptography. Він базується на симетричному ключі — тобто такому ключі, який використовується і для шифрування, і для розшифрування. Сам же ключ, за замовчуванням, зберігається у settings.py (див. Додаток). Це не є дуже гарною практикою — зберігати ключ в коді, оскільки, завантажуючи його, наприклад, на гітхаб, його можуть легко знайти, що може призвести до викрадення персональних даних. Тому такі ключі зберігаються в змінних середовища .env і витягуються з них у settings.py через інші методи, які спеціально на це налаштовані, в нашому випадку — функція config() з модуля decouple. Функції шифрування використовуються у models.py (див. Додаток) і є методами моделей [2] — таку можливість надає нам Django ORM — представлення таблиць бази даних як об'єкти Python.

2.3 Інтеграція сторонніх сервісів в систему

У рамках реалізації даного вебзастосунку використовується сторонній хмарний сервіс PythonAnywhere, який забезпечує зручне середовище для розгортання та публічного доступу до python-додатків [3]. Це дозволяє швидко опублікувати проєкт без потреби в оренді власного сервера.

Інтеграція з даним сервісом не потребує додаткового API, оскільки основна взаємодія з ним здійснюється через браузер або термінал: спочатку потрібно завантажити джанго-проєкт (максимальна вага — 512 Мб, треба стиснути проєкт у архів, наприклад, zip, і закинути його у сервіс у розділ Files (рисунок 2.3.1), оскільки просто так завантажити проєкт не вийде, потім його треба розархівувати через термінал bash console, вбудований у pythonanywhere командою `unzip *.zip` (рисунок 2.3.2)).

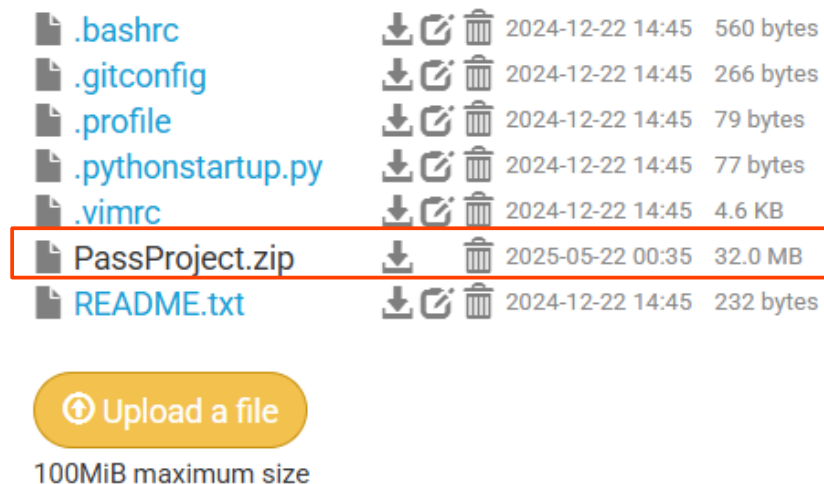


Рисунок 2.3.1 — Завантаження zip-проєкту на pythonanywhere

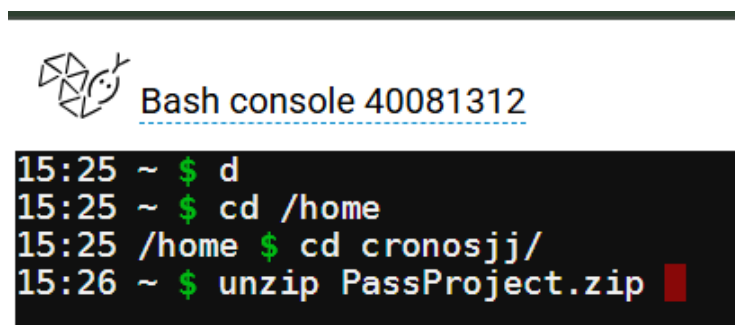


Рисунок 2.3.2 — Розархівування zip-проєкту

Тепер треба перейти у розділ Web і у підрозділі Code вказати абсолютний шлях до проєкту (рисунок 2.3.3), у підрозділі Static files вказати шлях до директорії, в якій знаходяться статичні файли (рисунок 2.3.4). Ці статичні файли [4] можна зібрати разом (user- та admin- static files), виконавши команду в терміналі:

```
python manage.py collectstatic
```

Code:

What your site is running.

Source code:	/home/cronosjj/PassProject/passwebbapp
Working directory:	/home/cronosjj/
WSGI configuration file:	/var/www/cronosjj_pythonanywhere_com_wsgi.py
Python version:	3.10 

Рисунок 2.3.3 — Встановлення абсолютного шляху до застосунку

Static files:

Files that aren't dynamically generated by your code, like CSS, JavaScript or uploaded files, can be served much faster straight off the disk if you specify them here. You need to **Reload your web app** to activate any changes you make to the mappings below.


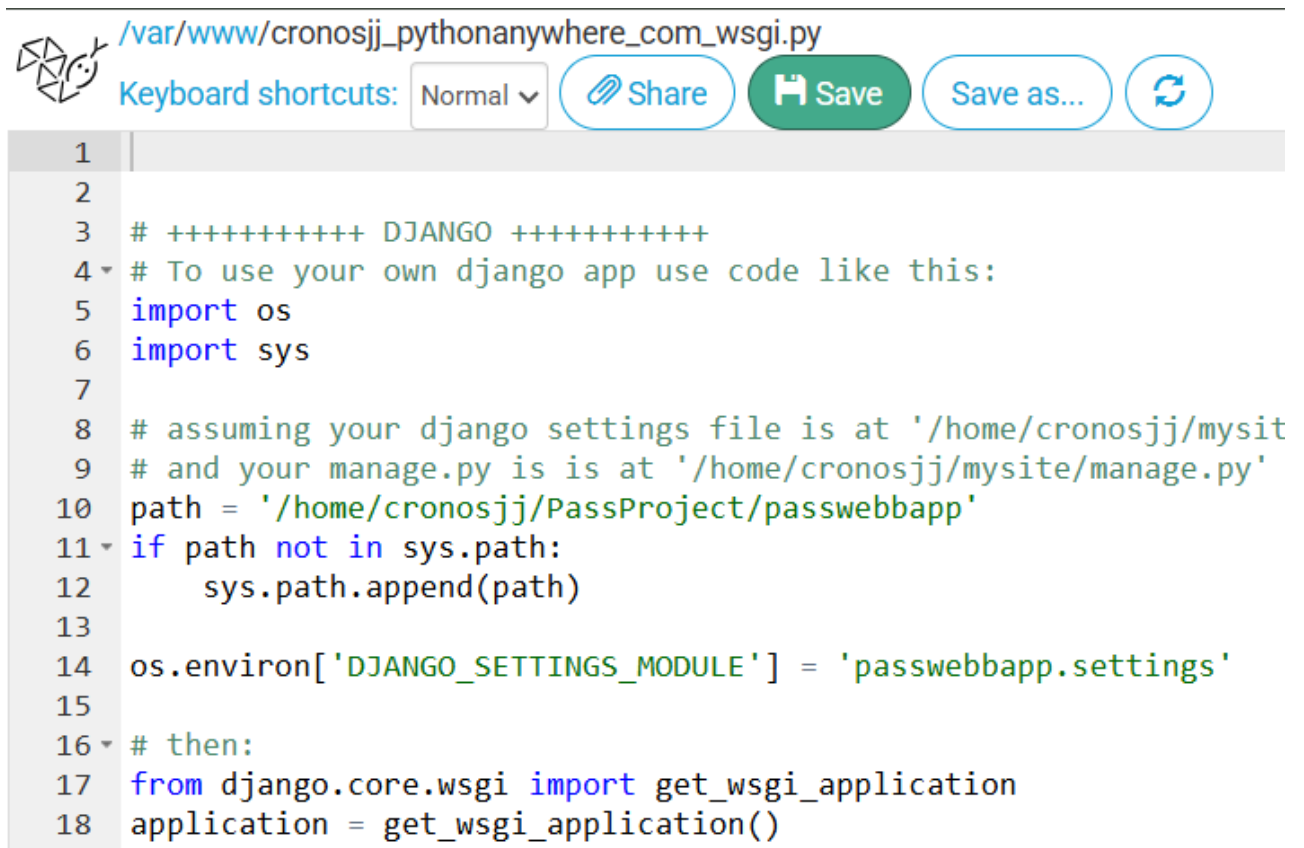
URL	Directory	Delete
/static/	/home/cronosjj/PassProject/passwebbapp/staticfiles	
Enter URL	Enter path	

Рисунок 2.3.4 — Встановлення абсолютного шляху до статичних файлів

Варто зазначити, що усі статичні файли, а саме файли стилів — style.css, файл скриптів — script.js, та картинки знаходяться у папці static, причому це вимога самого фреймворку. У базовому шаблоні base.html потрібно завантажувати ці файли через шаблонізатор jinja рядком {% load static %} [5].

Після цього потрібно відредагувати конфігураційний WSGI-файл так, щоб в ньому були налаштування тільки для django і коректні шляхи до відповідних

файлів і директорій (рисунк 2.3.5), змінити змінну у файлі settings.py для того, щоб надати дозвіл хостам користуватись цим додатком ALLOWED_HOSTS = ['*']; (рисунк 2.3.6), змінити змінну налаштувань DEBUG = False у змінних середовища .env і перезавантажити додаток у розділі Web→Reload.



```
1 2
2
3 # ++++++ DJANGO ++++++
4 # To use your own django app use code like this:
5 import os
6 import sys
7
8 # assuming your django settings file is at '/home/cronosjj/mysit
9 # and your manage.py is is at '/home/cronosjj/mysite/manage.py'
10 path = '/home/cronosjj/PassProject/passwebbapp'
11 if path not in sys.path:
12     sys.path.append(path)
13
14 os.environ['DJANGO_SETTINGS_MODULE'] = 'passwebbapp.settings'
15
16 # then:
17 from django.core.wsgi import get_wsgi_application
18 application = get_wsgi_application()
```

Рисунок 2.3.5 — Редагування конфігураційного WSGI-файлу

```
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = config("SECRET_KEY")

FERNET_KEY = config("FERNET_KEY")

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = config("DEBUG", cast=bool, default=False)

ALLOWED_HOSTS = ['*']
```

Рисунок 2.3.6 — Надання дозволу хостам користуватись застосунком

Існують певні недоліки використання цього сервісу, які відрізняються від платного хостингу: по-перше, можливо використовувати тільки один додаток,

тобто одночасно вивантажити на сервер два веб-додатка не вийде, по-друге, у розділі Web→Best before date потрібно час від часу оновлювати додаток, оскільки, якщо цього не робити протягом трьох місяців, то сайт перестане працювати на сервері, але у рамках даної роботи ці недоліки не відчуються.

2.4 Створення серверної архітектури додатку

Для розробки серверної частини обрано монолітну архітектуру, що дозволяє спростити розгортання та підтримку додатку на етапі розробки, а також зменшити складність управління додатком. Схема монолітної архітектури показана на рисунку 2.4.1:

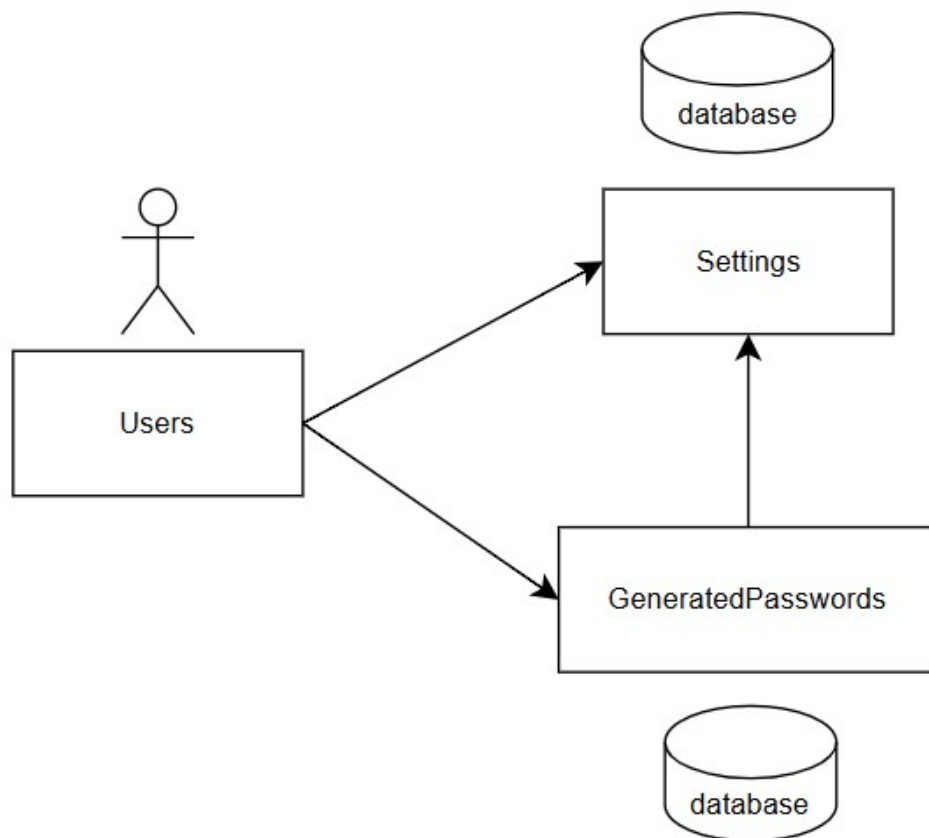


Рисунок 2.4.1 — Схема монолітної архітектури

Основні модулі системи:

1. Модуль користувачів (Users). Модуль користувачів відповідає за управління даними користувачів, включаючи реєстрацію, авторизацію та

зберігання інформації про користувачів, таких як електронна пошта, пароль тощо. Модуль взаємодіє з базою даних для збереження цих даних і використовується іншими модулями для аутентифікації та авторизації користувачів в системі.

2. Модуль налаштувань (Settings). Модуль налаштувань відповідає за параметри генерації паролів, зокрема довжину паролю, наявність цифр, літер та спеціальних символів. Він зберігає ці налаштування в базі даних і взаємодіє з іншими модулями, щоб забезпечити передачу налаштувань при генерації паролів.
3. Модуль генерації паролів (GeneratedPasswords). Цей модуль відповідає за безпечну генерацію паролів згідно з параметрами, вказаними в модулі налаштувань. Згенеровані паролі зберігаються в базі даних і асоціюються з конкретними користувачами. Модуль також надає API для доступу до згенерованих паролів.

Перевагою такого підходу є те, що усі модулі взаємодіють між собою напрямую, використовуючи спільну базу даних (хоча на схемі показано дві БД, насправді це одна й та сама база даних, це зроблено для того, щоб було зрозуміло, що обидва модулі Settings і GeneratedPasswords взаємодіють з базою даних), що дозволяє швидко організувати обмін даними між компонентами. Розгортання такого додатку потребує мінімуму налаштувань, що знижує технічні вимоги до сервера.

До недоліків можна віднести слабку масштабованість, тобто якщо навантаження зросте на якийсь один модуль, то змінювати потрібно буде усі модулі. Також складно впроваджувати нові технології для окремих частин проєкту — усі компоненти мають залишатися на одній платформі. Це трохи обмежує гнучкість системи та може сповільнювати розвиток проєкту в довгостроковій перспективі.

2.5 Реалізація обробки запитів та взаємодії з базою даних на сервері

Усі запити обробляються у `views.py`, причому яку саме функцію обробки використовувати вирішує `urls.py` – він дозволяє Django зіставляти вхідний запит із відповідним представленням, яке обробляє цей запит. Форми реєстрації, авторизації та генерації, що знаходяться у `forms.py` (див. Додаток) [6], використовуються у HTML-шаблонах [7], і при відправці цих форм на обробку серверу, функції `views.py` зчитують їх, обробляють і повертають певний результат у вигляді перенаправлення на якусь сторінку, або рендерингу нової сторінки.

Так, наприклад, головна функція усього проєкту — функція генерації паролю — викликається при відправці форми генерації користувачем, при натисканні кнопки «Згенерувати». Функція повертає пароль згідно з налаштуваннями, які йдуть як параметри функції, після чого функція обробки запиту повертає рендеринг сторінки, де в якості аргументу йде отриманий пароль. Таким чином користувач бачить його. У зареєстрованого користувача, окрім цього, цей пароль відправляється у базу даних. Це відбувається через створення самого поля: `generated_password = GeneratedPasswords(params)`, куди ми передаємо параметри у вигляді налаштування генерації, після чого звертаємось до методу створеного поля (об'єкта Python) `generated_password.save()`. В деяких інших методах обробки запитів, використовуються інші методи створення полів, наприклад, `get_or_create()` для налаштувань генерації при обробці запиту головної сторінки, оскільки, якщо такі налаштування вже є в БД, то створювати їх знову не потрібно.

У методах обробки історії паролів, головної сторінки, оновлення асоціацій до паролів (додавання та видалення) в обов'язковому порядку на початку витягується пошта користувача з сесій `request.session` і перевіряється чи існує такий користувач у базі даних через `Users.objects.filter(email=email).exists()`. Якщо ні, то його перенаправляють на сторінку входу в обліковий запис.

2.6 Тестування та налаштування веб додатку на сервері

Щоб протестувати застосунок на сервері, потрібно створити суперкористувача [2] в терміналі PyCharm командою:

```
python manage.py createsuperuser
```

Після встановлення імені та паролю для суперкористувача, ми можемо заходити в панель адміністратора і переглядати таблиці в базі даних та їх зміни в залежності від дій звичайних користувачів. Для цього треба перейти за посиланням <https://cronosjj.pythonanywhere.com/admin/>, заповнити дані і увійти в акаунт адміністратора (рисунок 2.6.1).

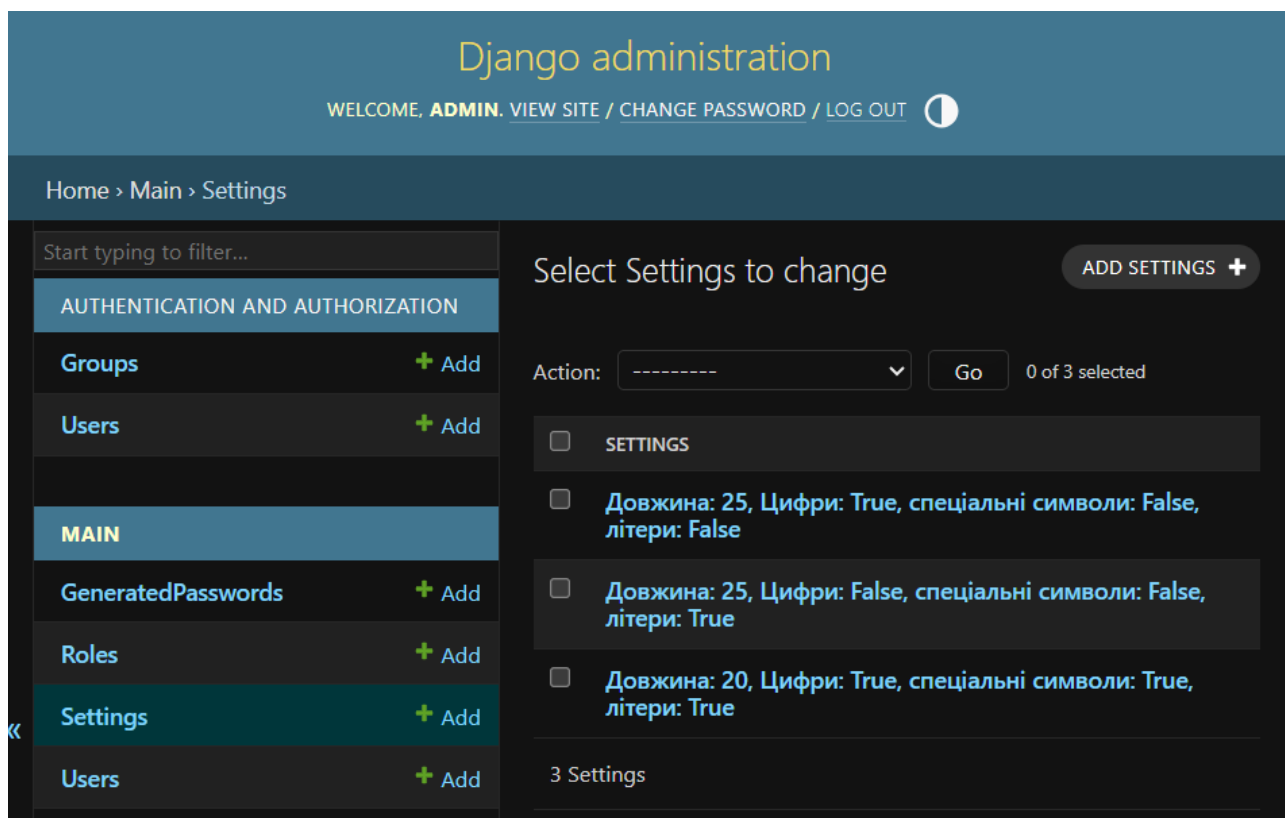


Рисунок 2.6.1 — Панель адміністратора

Тепер зі сторони звичайного користувача можемо, наприклад, зареєструватись і згенерувати декілька паролів. Створимо користувача з поштою `example@gmail.com` з паролем `1234` і подивимось, що у нас буде відбуватись з точки зору адміністратора (рисунок 2.6.2, рисунок 2.6.3).

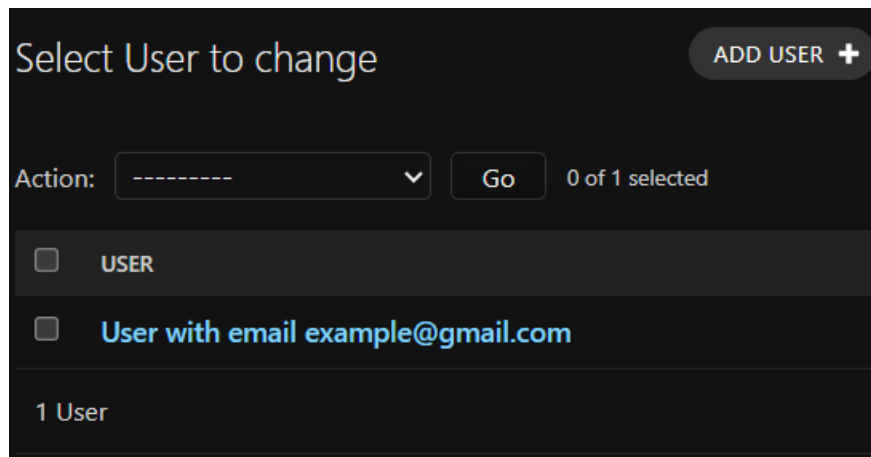


Рисунок 2.6.2 Таблиця — Users після створення акаунту користувачем

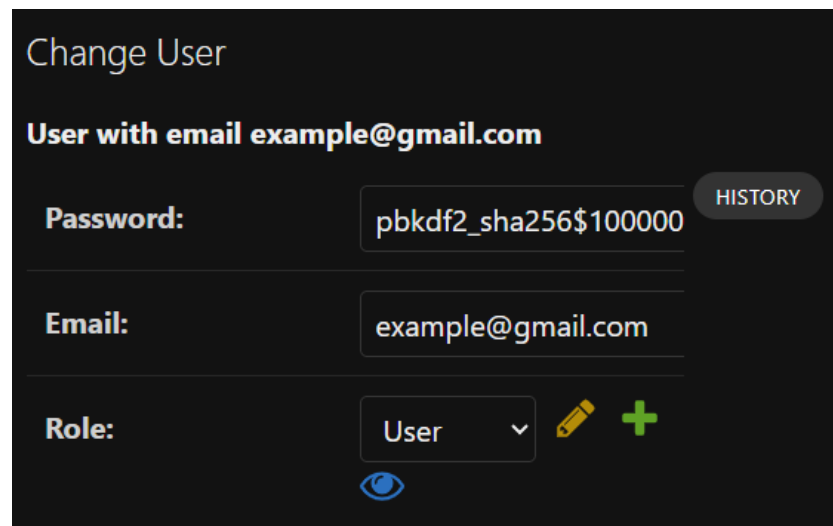


Рисунок 2.6.3 — Поле створеного користувача у таблиці Users

Як бачимо, такий користувач справді створився, а у полі пароля бачимо його хеш — тобто усе працює вірно. У полі Role бачимо User — встановлюється за замовчуванням. Тепер зайдемо в акаунт і згенеруємо декілька паролів та надамо їм певні асоціації (наприклад, пароль №1, пароль №2 та пароль №3). Знову подивимось на результат (рисунок 2.6.4). В таблиці GeneratedPasswords ми бачимо зашифровані паролі — так, як і задумано. Перейдемо по першому паролю, щоб впевнитись, що це створив саме той користувач, якого ми створили вище (рисунок 2.6.5). Як бачимо у нас у полі відображається «user with email example@gmail.com», тобто усе працює правильно.

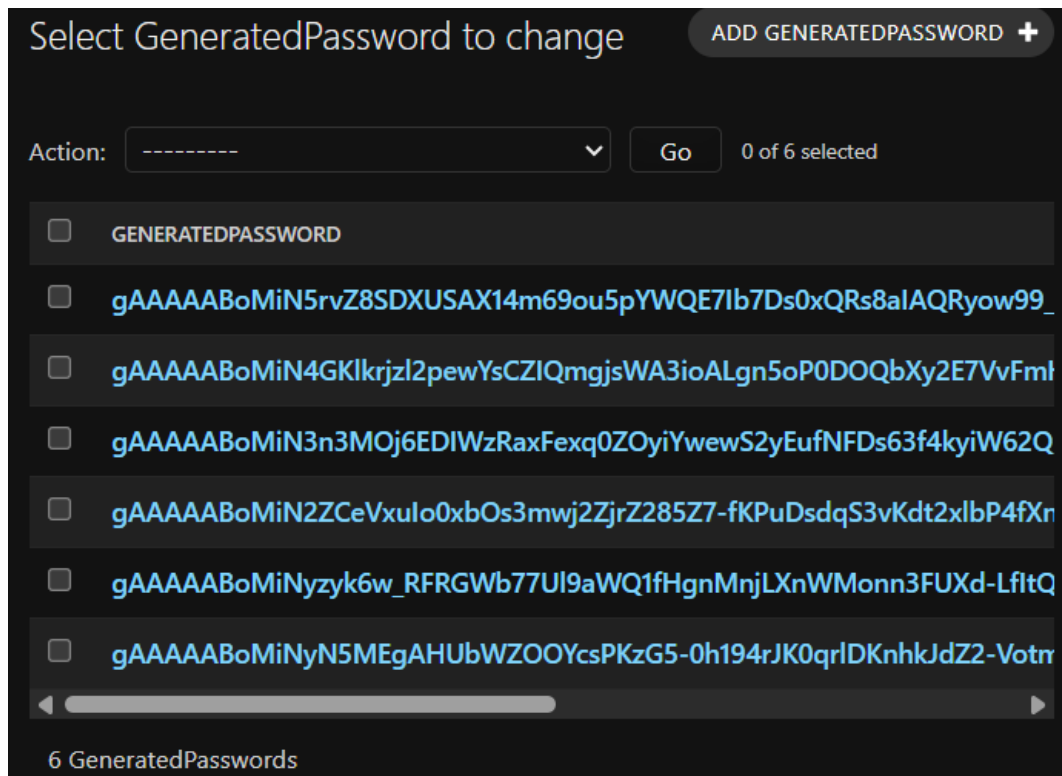


Рисунок 2.6.4 — Таблиця GeneratedPasswords після генерації паролів

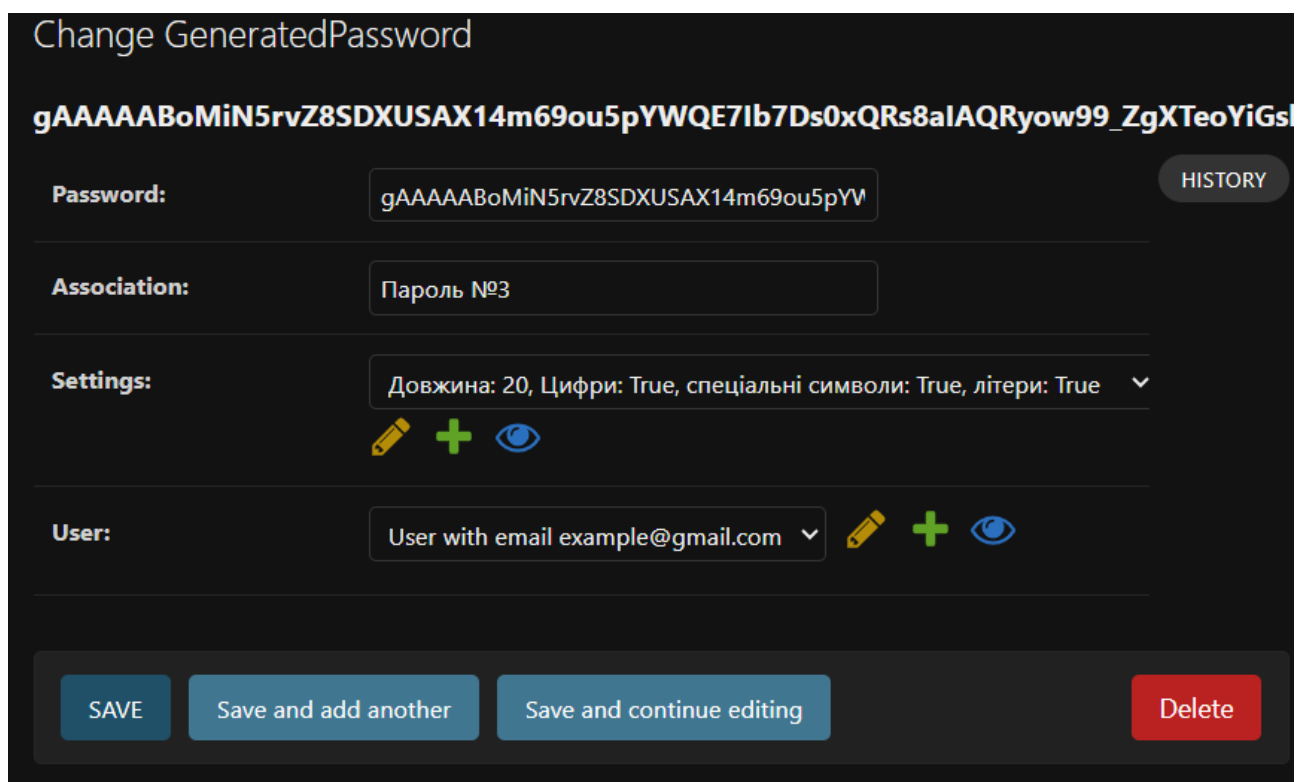


Рисунок 2.6.5 — Згенерований пароль в таблиці GeneratedPasswords

Для закріплення перевіримо налаштування паролів. Переходимо у таблицю Settings (рисунок 2.6.6).

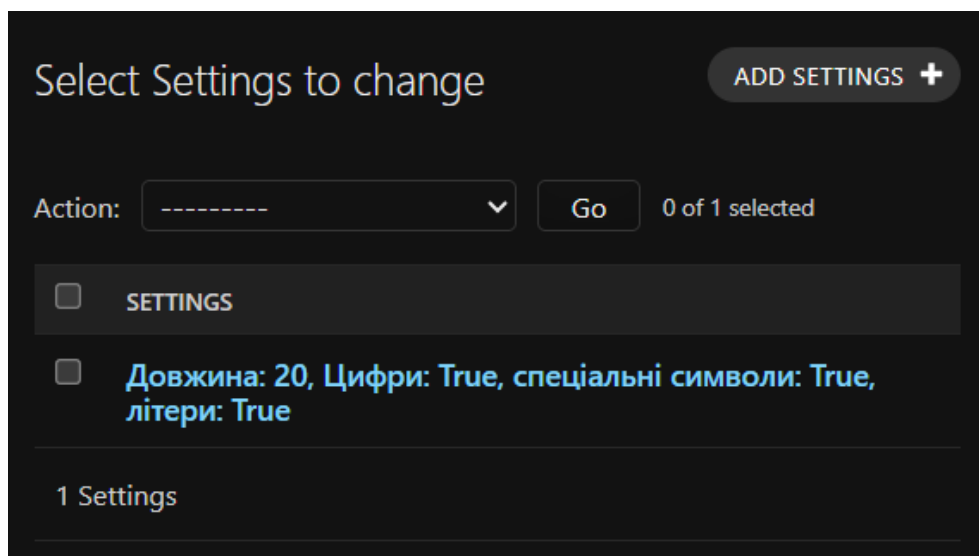


Рисунок 2.6.6 — Таблиця налаштувань після тестування

Бачимо ті ж самі налаштування, що й у таблиці GeneratedPasswords, оскільки саме через цю таблицю можна перевірити приналежність налаштувань паролів до користувача.

2.7 Функція генерації

Функція генерації реалізована наступним чином, на вхід ми приймаємо три булеві показники і довжину, далі ми перевіряємо кожну булеву змінну, якщо якась позитивна, то ми навмання вибираємо літеру з відповідної сукупності символів (цифри, літери, спеціальні символи). Це необхідно, щоб гарантовано в паролі був хоча б один символ з послідовності символів, яку вказує користувач. Потім циклом проходимось (умовою виходу буде різниця довжини і здвигу, зумовленого попередніми діями) і добираємо інші символи. Після цього перетасовуємо послідовність, оскільки перші три символи складають собою певний алгоритм, а нам потрібний абсолютний хаос.

Щоб зрозуміти, чому генерація справді надійна, потрібно трохи ознайомитись з фізичним змістом. І для вибору випадкових символів з

послідовності, і для їх перетасування використовуються методи з модуля `secrets` — модуль, призначений для генерації криптографічно безпечних випадкових чисел, саме випадкових чисел, бо звичайні модулі, наприклад `random`, використовують зерно для генерації, тобто існує певний алгоритм, за яким генерується послідовність, і тоді такі символи є псевдовипадковими. Аналогія в `unix`-системах, наприклад, це — функція `srand(time(NULL))`, де функція в якості зерна приймає кількість секунд, що пройшло з початку епохи UNIX – 1 січня 1970 р. І те і інше детерміноване, і якщо зловмисник якимось чином отримає зерно — фактично у нього буде ключ до усіх паролів, оскільки в такому випадку вони створюються за одними і тими ж правилами. Але `secrets` в своїй основі використовує `os.urandom(n)`, який в свою чергу використовує системні `BCryptGenRandom` у Windows, `/dev/urandom` — у Linux. Ці функції криптографічно стійкого генератора псевдовипадкових чисел і вони в свою чергу використовують системні джерела ентропії для генерування справжніх випадкових послідовностей байтів, які потім можна конвертувати в символи. Хоча в назві знову повторюється слово «псевдовипадковий», але ці генератори генерують такі послідовності символів, які не може відрізнити від абсолютно випадкових жоден алгоритм за поліноміальний час. Іншими словами, жоден статистичний тест не буде у змозі відрізнити отриману послідовність байтів від насправді випадкової послідовності. Це зумовлено тим, що він не блокується, якщо ентропія вичерпана, а використовує вже накопичену ентропію. Багато сучасних джерел ентропії базуються на фізичних процесах як-от електричний шум (тепловий шум у резисторі, шум напруги), тактильний або випадковий час (рух курсору миші, час між натисканнями клавіш). Формально положення курсора миші або час між натисканням клавіш представляють хаотичну систему, тобто таку систему яка все ж таки описується певним алгоритмом (нелінійними диференціальними рівняннями), але передбачити її неможливо, через гіперчутливість до початкових значень. Вперше таке питання постало за часів Ньютона, коли закон всесвітнього тяжіння добре описував рух двох

небесних тіл, але добавивши у систему третє тіло — і передбачення не збігалися з реальними даними. На початку 19 століття французький математик та фізик Анрі Пуанкаре довів, що така задача простого вирішення немає, а виною всьому — хаос. А ось електричний шум, тепловий шум, шум тунельного ефекту або зміни напруги в резисторах чи транзисторах є випадковим на фундаментальному рівні, оскільки тут втручається квантова механіка. Це випадковість, яка, як ми вважаємо, вбудована в саму природу (згідно з стандартною Копенгагенською інтерпретацією квантової механіки). Випадковість, яка не зумовлена нічим. Немає параметрів, які ми могли б підгледіти і передбачити стан системи, їх просто не існує. Вона просто констатує факт, що якась подія станеться протягом якогось періоду часу. Навіть якщо з ідеальною точністю відтворити систему (нескінчена знаків після коми) з точністю до кварків, то все одно результат буде непередбачуваний. І це сильно відрізняється від нашого звичного розуміння випадковості, наприклад, зустрілися два знайомі в магазині, які не бачились багато років, або підкидають монетку, або гральний кубик. В таких ситуаціях насправді нічого випадкового немає: якби ми знали певну кількість параметрів таких систем з задовільною точністю (на прикладі монетки в якості параметрів можуть бути початкова швидкість монети, кут нахилу, маса та розподіл маси монети, тобто її момент інерції, опір повітря, силу гравітації, тертя при кидку тощо), то майбутнє, точно так же, як і минуле, було б у нас просто перед очима, за умови, звісно, якщо вистачить обчислювальних ресурсів.

3 ІНСТРУКЦІЯ КОРИСТУВАЧА

При переході за посиланням <https://cronosjj.pythonanywhere.com/>, користувач потрапляє на головну сторінку веб-сайту PassGen (рисунок 3.1).

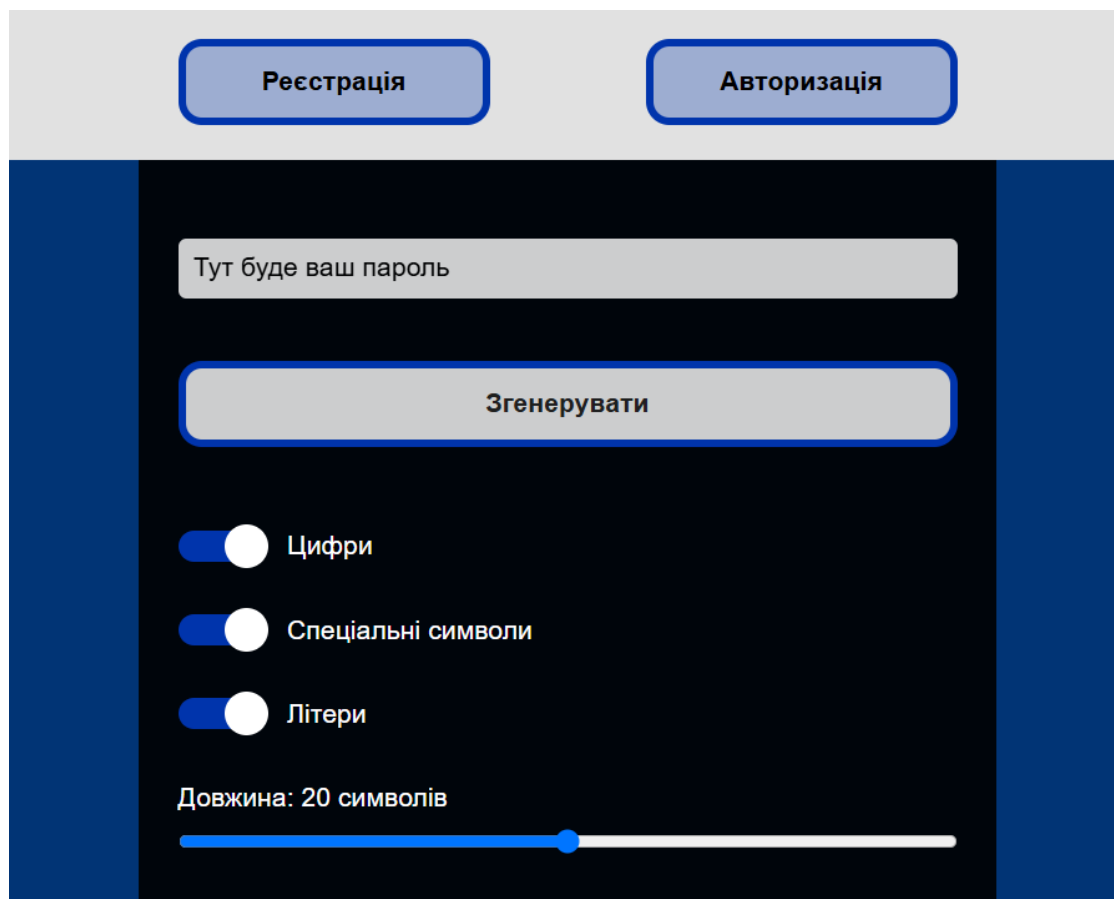


Рисунок 3.1 — Головна сторінка сайту

Зверху ми можемо побачити шапку сайту, де у користувача є можливість зареєструватися або увійти в акаунт. Далі знизу іде поле, де у користувача буде відображатися сам згенерований пароль, а під цим полем йде кнопка, при натисканні якої буде створено сам пароль. Далі ідуть налаштування генерації: перемикачі, при натисканні яких вони або включаються, або вимикаються, причому натискати можна не тільки на самі повзунки, а й на текст, що стоїть справа, і повзунок для задання довжини паролю, його можна зажати мишкою і перетаскувати зліва-направо (на мобільних пристроях перетаскувати пальцем) і відповідне значення довжини відображається трохи вище. Наприклад,

встановимо наступні налаштування: цифри — включені, спеціальні символи — вимкнені, літери — вимкнені, довжина — 25 символів, і натиснемо кнопку генерації паролю (рисунок 3.2).

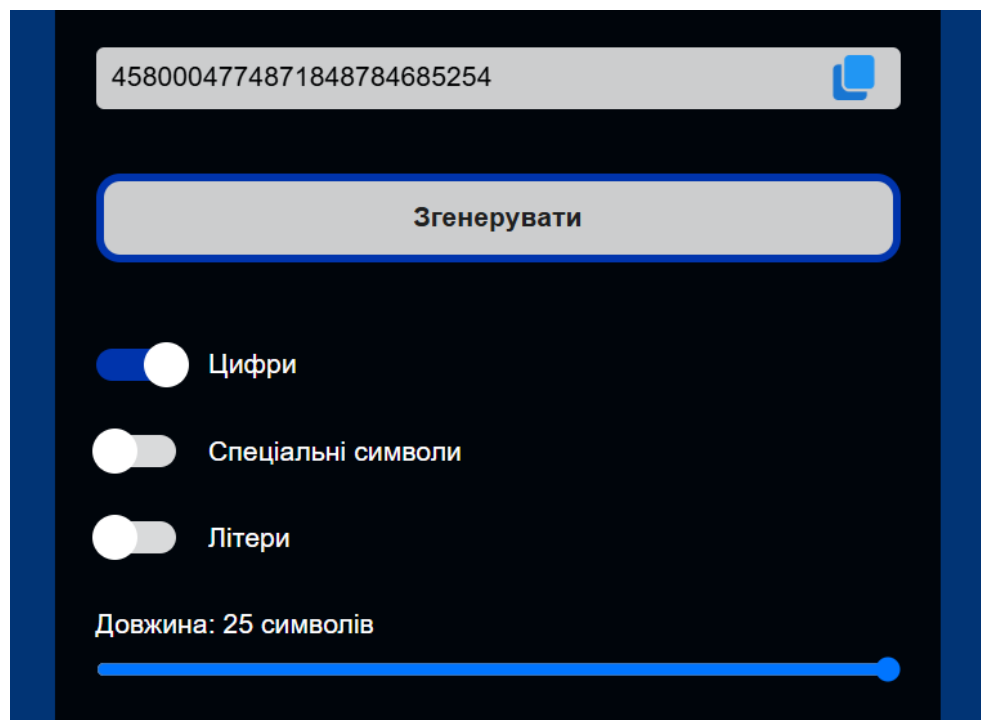


Рисунок 3.2 — Приклад генерації паролю

Після генерації паролю в полі створюється сам пароль, а в кінці поля з'являється невелика кнопка, натиснувши яку пароль копіюється в буфер обміну.

На цьому основний функціонал незареєстрованого користувача закінчується. Для отримання більших можливостей, потрібно зареєструватись. Для цього натискаємо кнопку «Реєстрація» і ми переходимо на сторінку реєстрації (рисунок 3.3).

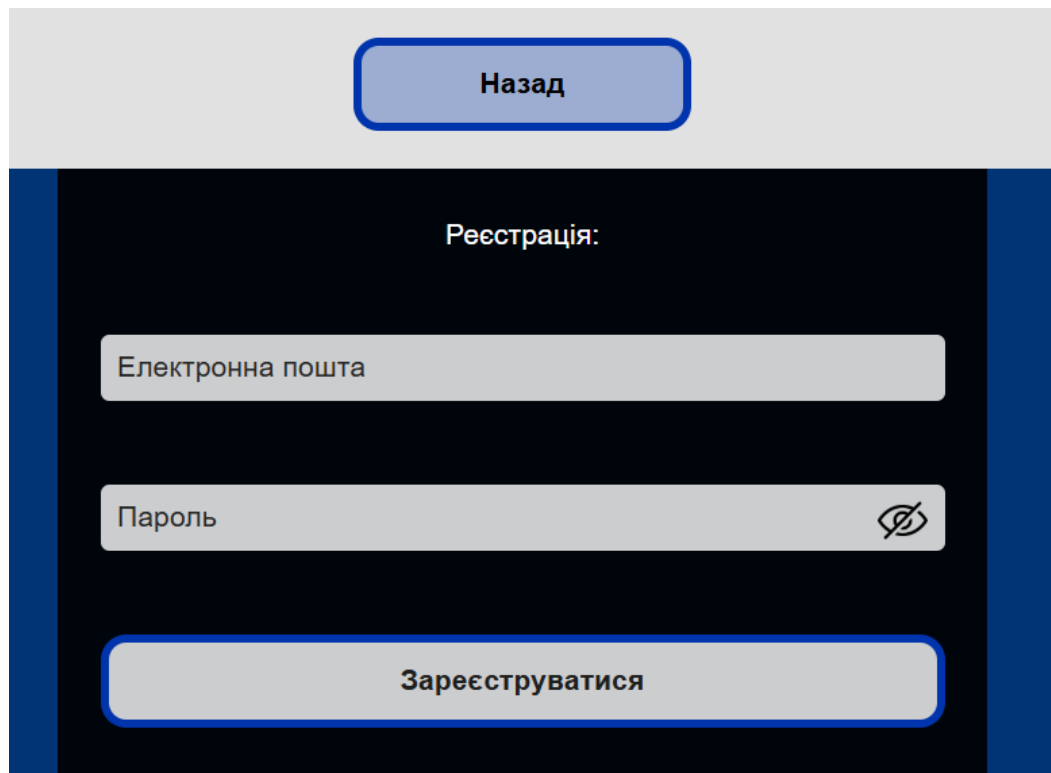
A screenshot of a registration form. At the top, there is a light gray header bar containing a blue button with the text "Назад". Below the header, the word "Реєстрація:" is centered. The form consists of two input fields: "Електронна пошта" and "Пароль". The "Пароль" field has a blue eye icon on its right side. Below the input fields is a large blue button with the text "Зареєструватися".

Рисунок 3.3 — Сторінка реєстрації

Тут зверху користувач може, при бажанні, повернутися на головну сторінку, натиснувши кнопку «Назад». Далі він може заповнити поля і, якщо щось не вірно, то в самому низу форми відобразиться помилка з відповідним повідомленням (рисунок 3.4).

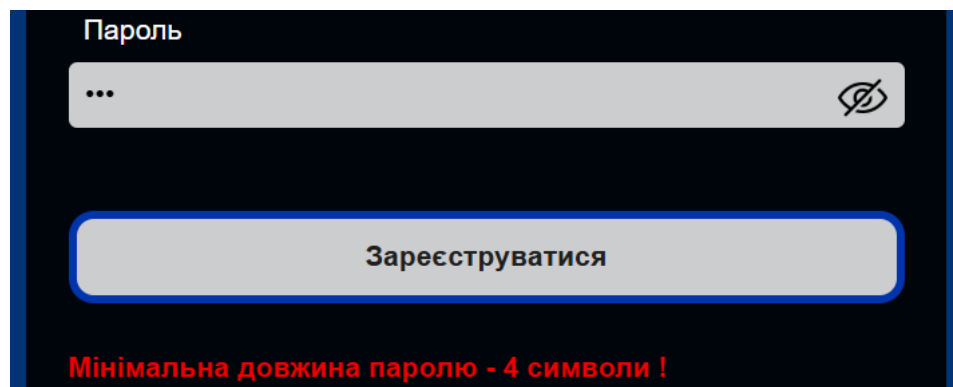
A screenshot of the registration form showing an error. The "Пароль" field is highlighted with a red border and contains three dots. Below the input fields is a large blue button with the text "Зареєструватися". At the bottom of the form, there is a red error message: "Мінімальна довжина паролю - 4 символи !".

Рисунок 3.4 — Помилка при реєстрації

У полі заповнення паролю в кінці є кнопка з іконкою перекресленого ока, при натисканні якої можна підглядіти те, що вводить користувач, і приховати в зворотному напрямку (рисунок 3.5).



Рисунок 3.5 — Підглядання введеного паролю

При успішній реєстрації, користувача перекидає на головну сторінку, де він може увійти у створений обліковий запис, натиснувши кнопку «Авторизація». Після цього його перенаправляють на сторінку авторизації (рисунок 3.6).

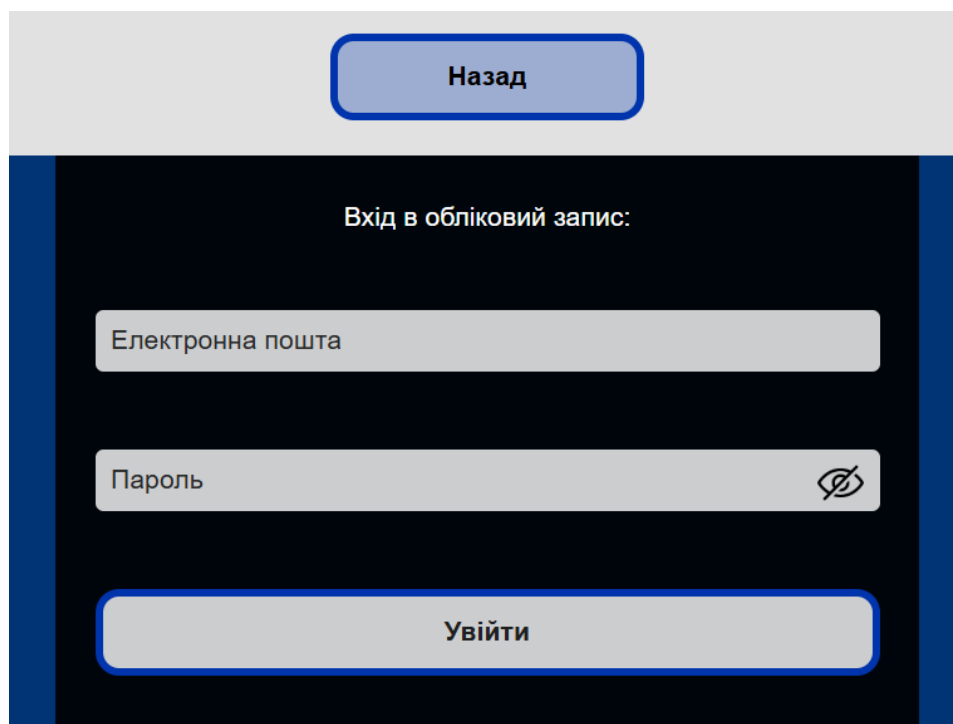


Рисунок 3.6 — Сторінка авторизації

Тут аналогічний дизайн і структура форми, як і при реєстрації: при введенні некоректних значень, або даних, яких немає в БД, тобто такого користувача не існує, або невірного паролю, то знизу форми буде виведено відповідну помилку червоного кольору. Також можна підглядіти введений пароль. Після натискання кнопки «Увійти», користувач потрапляє на головну сторінку з обліковим записом (рисунок 3.7).

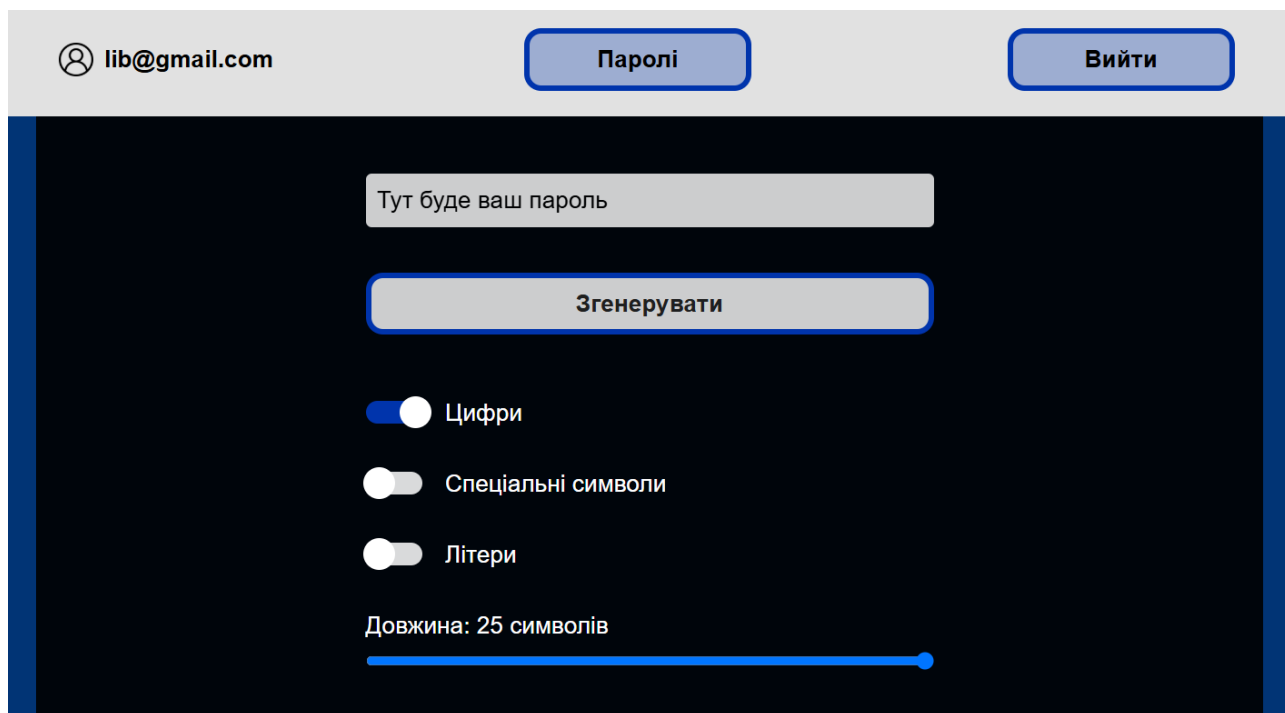


Рисунок 3.7 — Головна сторінка користувача

Зверху ми маємо трохи іншу шапку сайту: спочатку є інформація про користувача — іконка з поштою самого користувача, кнопка «Паролі», яка веде до сторінки з історією паролів (рисунок 3.8) і кнопка «Вийти» для виходу з облікового запису (після виходу користувач попадає на головну сторінку незареєстрованого користувача). Основна частина, яка йде під шапкою, така ж сама, як і на головній сторінці сайту, з однією тільки різницею: усі згенеровані паролі зберігаються і заносяться до секції історії паролів, тоді як у незареєстрованого користувача — ні.



Рисунок 3.8 — Сторінка історії паролів

Зверху знаходиться інформація про обліковий запис — іконка користувача з поштою (тут приклад, коли ширина екрану достатньо мала для виведення її на екран поруч з іконкою і щоб її побачити, потрібно навести мишкою на іконку, а на мобільних пристроях потрібно зажати пальцем іконку) та кнопка «Назад», при натисканні якої користувача повертає на головну сторінку з обліковим записом. Далі основна частина сторінки — список згенерованих паролів. Налаштування паролю відображається при наведенні на поле, де знаходиться сам пароль.

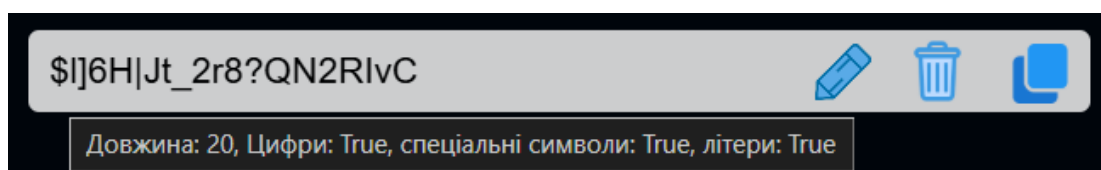


Рисунок 3.9 — Відображення налаштувань пароля

У кожному полі з паролем, справа користувач може побачити три кнопки з різними зображеннями: в кінці знаходиться кнопка копіювання (аналогічна кнопка, як і на сторінках генерації після створення паролю), посередині — кнопка видалення з зображенням кошика для сміття — видаляє поточний

пароль з історії, та кнопка додавання і редагування з іконкою олівця — додає опис до паролю. При наведенні на будь-яку кнопку, висвічується напис про те, що робить та, чи інша кнопка.

При натисканні на кнопку «Олівець» відкривається модальне вікно (рисунок 3.10).

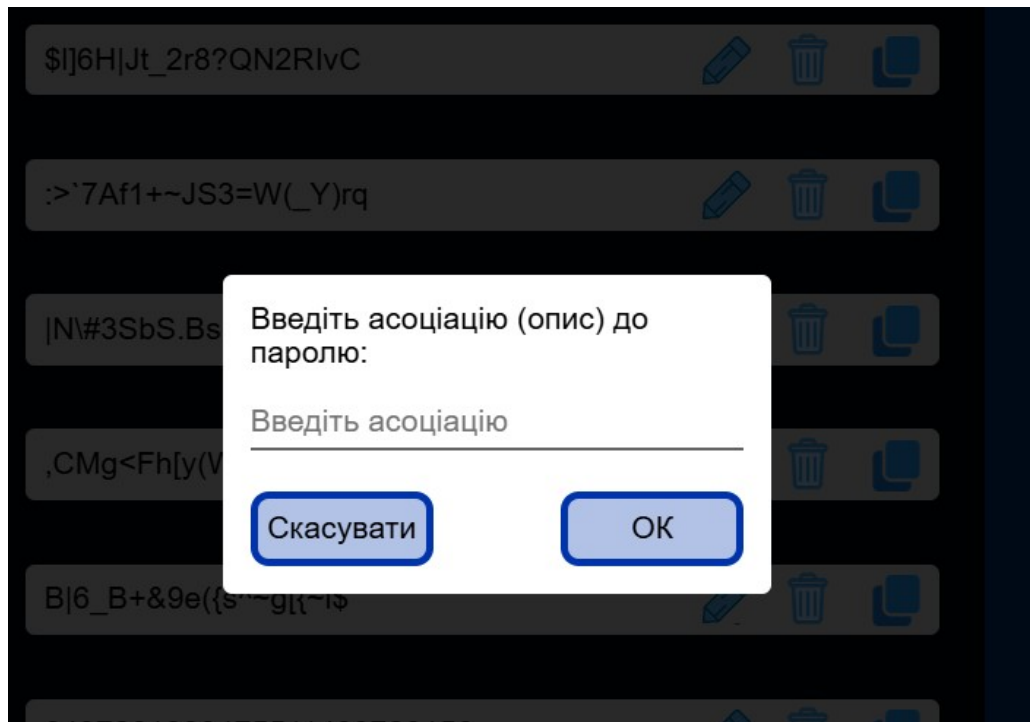


Рисунок 3.10 — Додавання асоціації до паролю

Користувачу надається можливість ввести опис, з яким у нього асоціюється пароль, у відповідне поле. Він може скасувати цю дію, натиснувши кнопку «Скасувати», або може заповнити поле вводу і натиснути кнопку «ОК» (рисунок 3.11).

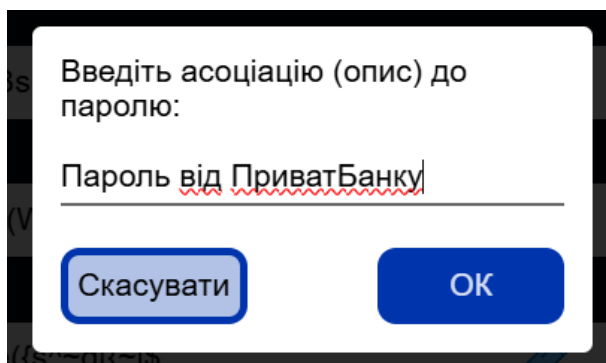


Рисунок 3.11 — Процес додавання асоціації до паролю

Після чого він одразу побачить зміни (рисунок 3.12).

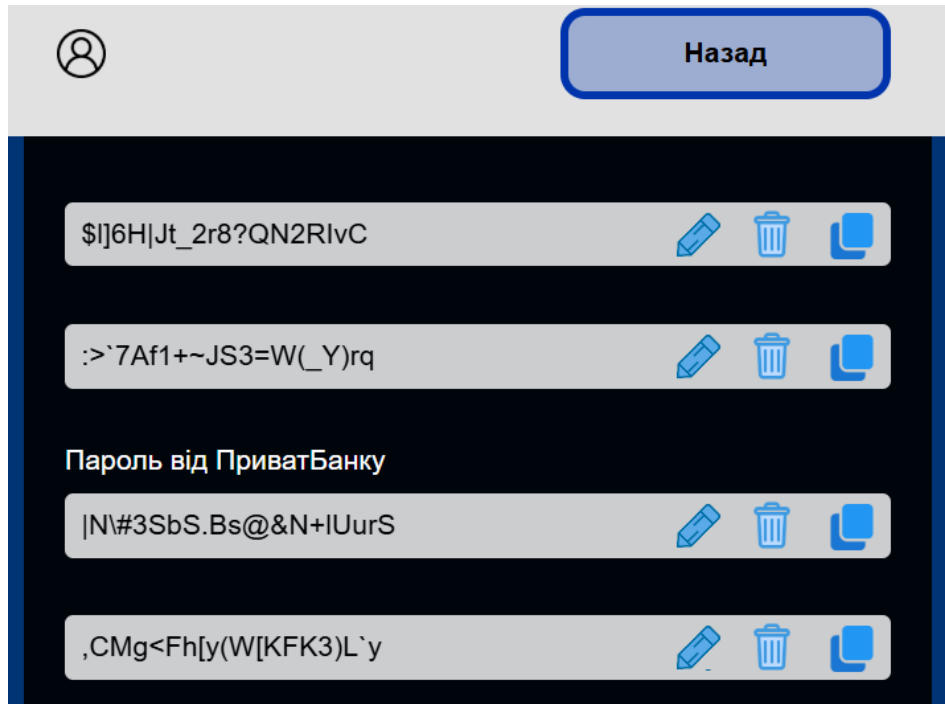


Рисунок 3.12 — Результат додавання опису до паролю

Якщо поле пусте або опис до паролю занадто великий, то знизу буде виведено відповідне попередження (рисунок 3.13).

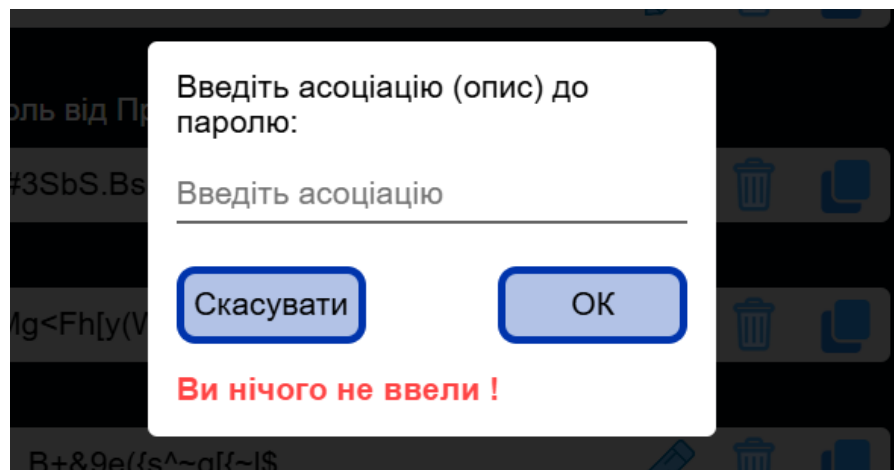


Рисунок 3.13 — Вивід помилки у модальному вікні

Для того щоб вийти з модального вікна, потрібно натиснути або на кнопку «Скасувати», або на пусте, затемнене місце поза самим вікном.

ВИСНОВКИ

У ході виконання цієї курсової роботи було розроблено веб-застосунок для генерації надійних паролів. Було використано знання, отримані під час вивчення дисципліни «Основи веб-програмування». Це дозволило не лише закріпити базові теоретичні знання на практиці, а й розширити власне уявлення про сферу веб-розробки.

Отриманий результат може бути використаний користувачами для створення стійких до зламу паролів із подальшим їх використанням для захисту персональної та конфіденційної інформації.

Для реалізації застосунку було обрано фреймворк Django, який повністю задовольнив потреби розробки — від створення HTML-шаблонів до обробки запитів на сервері. Як середовище розробки використовувався PyCharm Community, однак воно виявилось не зовсім зручним: форматування коду на JavaScript та CSS не працювало належним чином. Наскільки вдалося з'ясувати, відповідні плагіни доступні лише у платній версії середовища. У подальшому варто спробувати реалізацію подібних проєктів в інших IDE та редакторах, наприклад, Visual Studio або VS Code.

Подальший розвиток проєкту може включати усунення деяких недоліків. Зокрема, кожне відправлення запиту на сервер наразі супроводжується повним перезавантаженням сторінки за допомогою функцій `redirect()` та `render()`. Це може викликати візуальні затримки й створювати не зовсім приємне користувацьке враження. Крім того, доцільним буде перенесення застосунку на платний хостинг із власним доменним ім'ям — для покращення індексації сайту в пошукових системах та підвищення загального рівня довіри користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Django documentation: password management in Django. URL:
<https://docs.djangoproject.com/en/5.2/topics/auth/passwords/#increasing-thesalt-entropy> (дата звернення 24.05.2025).
2. Django documentation: Part 2: Models and the admin site. URL:
<https://docs.djangoproject.com/en/5.2/intro/tutorial02/> (дата звернення 24.05.2025).
3. Deploying an existing Django project on PythonAnywhere. URL:
<https://help.pythonanywhere.com/pages/DeployExistingDjangoProject/> (дата звернення: 24.05.2025).
4. How to setup static files in Django. URL:
<https://help.pythonanywhere.com/pages/DjangoStaticFiles> (дата звернення: 24.05.2025).
5. Django documentation: Part 6: Static files. URL:
<https://docs.djangoproject.com/en/5.2/intro/tutorial06/> (дата звернення 25.05.2025).
6. Django documentation: Part 4: Forms and generic views. URL:
<https://docs.djangoproject.com/en/5.2/intro/tutorial04/> (дата звернення 25.05.2025).
7. Django documentation: Part 3: Views and templates. URL:
<https://docs.djangoproject.com/en/5.2/intro/tutorial03/> (дата звернення 25.05.2025).

ДОДАТОК

Код програми

НТУУ «КПІ ім. І. Сікорського» НН ІАТЕ ТВ-33

Листів 28

Київ — 2025

admin.py

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

admin.site.register(models.Roles)
admin.site.register(models.Settings)
admin.site.register(models.Users)
admin.site.register(models.GeneratedPasswords)
```

forms.py

```
from django.forms import (Form, EmailInput, PasswordInput,
IntegerField, NumberInput, TextInput,
                        CharField, EmailField, BooleanField,
CheckboxInput, HiddenInput)

class LoginForm(Form):
    email = EmailField(widget=EmailInput(attrs={
        'id': 'email', 'class': 'form-control', 'placeholder':
        'Електронна пошта',
    }))
    password = CharField(widget=PasswordInput(attrs={
        'id': 'password', 'class': 'form-control', 'placeholder':
        'Пароль',
    }))

class RegistrationForm(Form):
    email = EmailField(widget=EmailInput(attrs={
        'id': 'email', 'class': 'form-control', 'placeholder':
        'Електронна пошта',
    }))
    password = CharField(widget=PasswordInput(attrs={
        'id': 'password', 'class': 'form-control', 'placeholder':
        'Пароль',
    }))

class GenerateForm(Form):
    is_numbers = BooleanField(required=False, initial=True,
widget=CheckboxInput())
    is_special = BooleanField(required=False, initial=True,
widget=CheckboxInput())
    is_letters = BooleanField(required=False, initial=True,
widget=CheckboxInput())
    length = IntegerField(min_value=15, max_value=25, initial=20,
        widget=NumberInput(attrs={'type': 'range', 'step': '1',
        'class': 'length-range'}))
    )
```

models.py

```
from django.contrib.auth.hashers import make_password,
check_password
from django.db import models
from django.conf import settings
from cryptography.fernet import Fernet
```



```

class Roles(models.Model):
    roleName = models.CharField(max_length=255)

    def __str__(self):
        return f"{self.roleName}"

    class Meta:
        verbose_name = "Role"
        verbose_name_plural = "Roles"

class Users(models.Model):
    password = models.CharField(max_length=255)
    email = models.CharField(max_length=255, unique=True)
    role = models.ForeignKey(Roles, on_delete=models.CASCADE)

    def set_password(self, password):
        self.password = make_password(password)

    def check_password(self, password):
        return check_password(password, self.password)

    def __str__(self):
        return f"User with email {self.email}"

    class Meta:
        verbose_name = "User"
        verbose_name_plural = "Users"

class Settings(models.Model):
    length = models.IntegerField()
    isNumbers = models.BooleanField()
    isLetters = models.BooleanField()
    isSpecial = models.BooleanField()

    def __str__(self):
        return (f"Довжина: {self.length}, Цифри: {self.isNumbers}"
                f", спеціальні символи: {self.isSpecial}, літери: {self.isLetters}")

    class Meta:
        verbose_name = "Settings"
        verbose_name_plural = "Settings"

class GeneratedPasswords(models.Model):
    password = models.CharField(max_length=255)
    association = models.CharField(max_length=255)

```

```

        settings = models.ForeignKey(Settings,
on_delete=models.CASCADE)

        user = models.ForeignKey(Users, on_delete=models.CASCADE)

    def __str__(self):
        return f"{self.password}"

    def set_password(self, password):
        f = Fernet(settings.FERNET_KEY)
        self.password = f.encrypt(password.encode()).decode()

    def get_password(self):
        f = Fernet(settings.FERNET_KEY)
        return f.decrypt(self.password.encode()).decode()

    class Meta:
        verbose_name = "GeneratedPassword"
        verbose_name_plural = "GeneratedPasswords"

```

main.urls.py

```

from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('registration', views.registration, name='registration'),
    path('login', views.login, name='login'),
    path('home', views.home, name='home'),
    path('logout', views.logout, name='logout'),
    path('history', views.history, name='history'),
    path('delete/<int:pk>/', views.delete_password,
name='delete'),
    path('update/', views.update_association, name='update')
]

```

views.py

```

from django.shortcuts import render, redirect
from .forms import RegistrationForm, LoginForm, GenerateForm
from .models import Users, Settings, GeneratedPasswords, Roles

import secrets
import string

def generate_password(is_numbers, is_special, is_letters, length):
    password = ''
    characters = ''
    shift = 0

    if is_numbers:
        characters += string.digits
        random_character = secrets.choice(string.digits)
        password += random_character

```

```

        shift += 1
    if is_letters:
        characters += string.ascii_letters
        random_character = secrets.choice(string.ascii_letters)
        password += random_character
        shift += 1
    if is_special:
        characters += string.punctuation
        random_character = secrets.choice(string.punctuation)
        password += random_character
        shift += 1

    i = 0
    while i < length - shift:
        random_char = secrets.choice(characters)
        password += random_char
        i += 1
    password = list(password)
    secrets.SystemRandom().shuffle(password)
    password = ''.join(password)
    return password

def set_form_values_from_request_session(request, form):
    for key_name in ('is_numbers', 'is_special', 'is_letters',
                     'length'):
        if request.session.get(key_name) is not None:
            form.fields[key_name].initial =
request.session.get(key_name)

def get_input_values_from_request_session(request, form):
    inputs = []
    for key_name in ('is_numbers', 'is_special', 'is_letters',
                     'length'):
        request.session[key_name] = form.cleaned_data[key_name]
        inputs.append(request.session.get(key_name))
    return tuple(inputs)

def index(request):
    password_output = 'Тут буде ваш пароль'
    if request.method == 'POST':
        form = GenerateForm(request.POST)
        if form.is_valid():
            is_numbers, is_special, is_letters, length = (
                get_input_values_from_request_session(request,
form)
            )
            if is_numbers is False and is_special is False and
is_letters is False:
                return render(request, 'index.html', {
                    'form': form,
                    'error': 'Виберіть хоча б якісь знаки !',
                    'password': password_output,

```

```

        })
        password_output = generate_password(is_numbers,
is_special, is_letters, length)
        return render(request, 'index.html', {
            'form': form, 'password': password_output,
        })
        form = GenerateForm()
        set_form_values_from_request_session(request, form)
        return render(request, 'index.html', {
            'form': form, 'password': password_output,
        })

def home(request):
    email = request.session.get('email')
    if not email or not
Users.objects.filter(email=email).exists():
        return redirect('login')
    password_output = 'Тут буде ваш пароль'
    if request.method == 'POST':
        form = GenerateForm(request.POST)
        if form.is_valid():
            is_numbers, is_special, is_letters, length = (
                get_input_values_from_request_session(request,
form)
            )
            if is_numbers is False and is_special is False and
is_letters is False:
                return render(request, 'home.html', {
                    'form': form,
                    'error': 'Виберіть хоча б якісь знаки !',
                    'password': password_output,
                    'email': email
                })
            password_output = generate_password(is_numbers,
is_special, is_letters, length)
            settings, created = Settings.objects.get_or_create(
                length=length, isLetters=is_letters,
isSpecial=is_special, isNumbers=is_numbers
            )
            generated_password = GeneratedPasswords(
                user=Users.objects.get(email=email),
settings=settings, association=""
            )
            generated_password.set_password(password_output)
            generated_password.save()
            return render(request, 'home.html', {
                'form': form, 'password': password_output,
'email': email
            })
            form = GenerateForm()
            set_form_values_from_request_session(request, form)
            return render(request, 'home.html', {

```

```

        'form': form, 'password': password_output, 'email': email
    })

def registration(request):
    form = RegistrationForm()
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            email = form.cleaned_data['email']
            password = form.cleaned_data['password']
            if Users.objects.filter(email=email).exists():
                return render(request, 'registration.html',
                              {'form': form,
                               'error': 'Користувач з такою поштою
уже існує'})
            role, created =
Roles.objects.get_or_create(roleName="User")
            role.save()
            user = Users(email=email, role=role)
            user.set_password(password)
            user.save()
            return redirect('/')

        return render(request, 'registration.html', { 'form': form, })

def login(request):
    form = LoginForm()
    if request.method == 'POST':
        form = LoginForm(request.POST)
        if form.is_valid():
            email = form.cleaned_data['email']
            password = form.cleaned_data['password']
            if Users.objects.filter(email=email).exists():
                user = Users.objects.get(email=email)
                if user.check_password(password):
                    request.session['email'] = email
                    return redirect('home',)
            return render(request, 'login.html',
                          {'form': form,
                           'error': 'Неправильна пошта або
пароль'})

        return render(request, 'login.html', { 'form': form, })

def logout(request):
    request.session.flush()
    return redirect('/')

def history(request):
    email = request.session.get('email')
    if not email or not
Users.objects.filter(email=email).exists():

```

```

        return redirect('login')
    passwords =
GeneratedPasswords.objects.filter(user=Users.objects.get(email=email))
    enc_passwords = []
    for password in passwords:
        enc_passwords.append((password.id,
password.get_password(), password.association, password.settings))
    return render(request, 'history.html', {
        'email': email, 'passwords': enc_passwords
    })

def delete_password(request, pk):
    email = request.session.get('email')
    if not email or not
Users.objects.filter(email=email).exists():
        return redirect('login')
    if GeneratedPasswords.objects.filter(id=pk,
user=Users.objects.get(email=email)).exists():
        GeneratedPasswords.objects.get(id=pk).delete()
    return redirect('history')

def update_association(request):
    email = request.session.get('email')
    if not email or not
Users.objects.filter(email=email).exists():
        return redirect('login')
    association = request.POST.get('association')
    password_id = request.POST.get('password-id')
    password = GeneratedPasswords.objects.get(id=password_id)
    password.association = association
    password.save()
    return redirect('history')

```

passwebbapp.urls.py

```

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

from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path("admin/", admin.site.urls),
    path("", include("main.urls")),
] + static(settings.STATIC_URL,
document_root=settings.STATIC_ROOT)

```

settings.py

```

from pathlib import Path
from decouple import config
import os

```

```

BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = config("SECRET_KEY")

FERNET_KEY = config("FERNET_KEY")

DEBUG = config("DEBUG", cast=bool, default=False)

ALLOWED_HOSTS = ['*']

INSTALLED_APPS = [
    "main",
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    'django_extensions',
]

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",
]

ROOT_URLCONF = "passwebbapp.urls"

TEMPLATES = [
    {
        "BACKEND":
"django.template.backends.django.DjangoTemplates",
        "DIRS": [],
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",

"django.contrib.messages.context_processors.messages",
            ],
        },
    ],
]

WSGI_APPLICATION = "passwebbapp.wsgi.application"

```

```

DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.sqlite3",
        "NAME": 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_TZ = True

STATIC_URL = "/static/"
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
STATICFILES_DIRS = [
    BASE_DIR / "main/static",
]

DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"

```

base.html

```

{% load static %}
<!doctype html>
<html lang="uk">
<head>
    <meta charset="UTF-8">

```



```

    <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <link rel="icon" href="{% static 'img/favicon.png' %}">
    <link rel="stylesheet" href="{% static 'css/style.css' %}">
    <title>PassGen</title>
</head>
<body>
    <header>
        <div class="header-content">
            {% block header %}
            {% endblock %}
        </div>
    </header>
    <div class="container">
        {% block content %}
        {% endblock %}
    </div>
    <script src="{% static 'js/script.js' %}"></script>
</body>
</html>

```

generation.html

```

<div class="generation">
    <form method="POST" class="generation-form" id="g-form"
novalidate>
        {% csrf_token %}
        <div class="generation-form__main">
            <div class="input-password">
                <input type="text" value="{{ password }}"
id="password-id" class="form-control" readonly>
                {% if password and password != "Тут буде ваш
пароль"%}
                    <button title="Скопіювати пароль" type="reset"
class="copy-btn"
onclick="copyTextPassword('password-id')"></button>
                {% endif %}
            </div>
            <button type="submit" class="registration-login-
button">Згенерувати</button>
        </div>
        <div class="generation-form__settings">
            <div class="checkbox">
                Цифри
                {{ form.is_numbers }}
            </div>
            <div class="checkbox">
                Спеціальні символи
                {{ form.is_special }}
            </div>
            <div class="checkbox">

```

```

        Літери
        {{ form.is_letters }}
    </div>
    <div>
        <p>Довжина: <span id="range-value"></span>
СИМВОЛІВ</p>
        {{ form.length }}
    </div>
</div>
<p class="registration-login-form-error">{{ error }}</p>
</form>
</div>

```

history.html

```

{% extends 'base.html' %}
{% load static %}

{% block header %}
    <div class="email-title">
        
        <p title="{{ email }}">{{ email }}</p>
    </div>
    <a href="{% url 'home' %}"
class="header-button"><p>Назад</p></a>
{% endblock %}

{% block content %}
    <div class="history">
        {% for password in passwords %}
            <div class="history-item">
                <div class="history-row">
                    <span>{{ password.2 }}</span>
                </div>
                <div class="input-password"
title="{{ password.3 }}">
                    <input type="text" value="{{ password.1 }}"
id="password-id-{{password.0}}" class="form-control" readonly>
                    <button title="Скопіювати пароль" type="reset"
class="copy-btn"
                        onclick="copyTextPassword('password-
id-{{password.0}}')"></button>
                    <a href="{% url 'delete' password.0 %}"
class="delete-btn" title="Видалити пароль"
onclick="setScrollToSession()"></a>
                    <button title="Додати/змінити асоціацію"
type="reset" class="update-btn"
onclick="openPopup('{{ password.0 }}')"></button>
                </div>
            </div>
        {% endfor %}
    </div>

```

```

        {% endfor %}
        <div class="popup" id="popup">
            <div class="popup__body">
                <form method="POST" action="{% url 'update' %}"
class="popup__content">
                    {% csrf_token %}
                    <p>Введіть асоціацію (опис) до паролю:</p>
                    <input type="hidden" name="password-id"
id="password-id-field">
                    <input type="text" name="association"
id="association-field" placeholder="Введіть асоціацію">
                    <div class="confirm-buttons">
                        <button type="button" id="cancel-button"
onclick="closePopup()" tabindex="2">Скасувати</button>
                        <button type="submit" tabindex="1"
onclick="setScrollToSession()">OK</button>
                    </div>
                    <p id="confirm-error"></p>
                </form>
            </div>
        </div>
    </div>
{% endblock %}

```

home.html

```

{% extends 'base.html' %}
{% load static %}

{% block header %}
    <div class="email-title">
        
        <p title="{{ email }}">{{ email }}</p>
    </div>
    <a href="{% url 'history' %}" class="header-
button"><p>Паролі</p></a>
    <a href="{% url 'logout' %}"
class="header-button"><p>Вийти</p></a>
{% endblock %}

{% block content %}
    {% include 'generation.html' %}
{% endblock %}

```

index.html

```

{% extends 'base.html' %}
{% load static %}

{% block header %}
    <a href="{% url 'registration' %}" class="header-
button"><p>Реєстрація</p></a>

```

```

    <a href="{% url 'login' %}" class="header-
button"><p>Авторизація</p></a>
{% endblock %}

```

```

{% block content %}
    {% include 'generation.html' %}
{% endblock %}

```

login.html

```

{% extends 'base.html' %}
{% load static %}

{% block header %}
    <a href="{% url 'index' %}" class="header-button reg-log-
button-back"><p>Назад</p></a>
{% endblock %}

{% block content %}
    <div class="registration-login-text">Вхід в обліковий
запис:</div>
    <form method="POST" id="l-form" class="registration-login-
form" novalidate>
        {% csrf_token %}
        {{ form.email }}
        <div class="password-container">
            {{ form.password }}
            
        </div>
        <button class="registration-login-button"
type="submit">Увійти</button>
        <p class="registration-login-form-error">{{ error }}</p>
    </form>
{% endblock %}

```

registration.html

```

{% extends 'base.html' %}
{% load static %}

{% block header %}
    <a href="{% url 'index' %}" class="header-button reg-log-
button-back"><p>Назад</p></a>
{% endblock %}

{% block content %}
    <div class="registration-login-text">Реєстрація:</div>
    <form method="POST" id="r-form" class="registration-login-
form" novalidate>
        {% csrf_token %}
        {{ form.email }}
        <div class="password-container">
            {{ form.password }}

```

```

        
    </div>
    <button class="registration-login-button"
type="submit">Записаться</button>
    <p class="registration-login-form-error">{{ error }}</p>
</form>
{% endblock %}

```

script.js

```

function togglePassword() {
    const eyeIcon = document.querySelector('.eye-icon');
    const passwordInput = document.getElementById('password');
    if (passwordInput.type == "password") {
        passwordInput.type = "text";
        eyeIcon.src = "static/img/visible.png";
    } else {
        passwordInput.type = "password";
        eyeIcon.src = "static/img/hide.png";
    }
}

function copyTextPassword(idPassword) {
    let copyText = document.getElementById(idPassword);
    copyText.select();
    copyText.setSelectionRange(0, 30);
    navigator.clipboard.writeText(copyText.value)
    window.getSelection().removeAllRanges();
}

function openPopup(passwordId, currentAssociation) {
    document.getElementById("password-id-field").value =
passwordId;

    header = document.querySelector('header');
    const scrollbarWidth = window.innerWidth - header.offsetWidth;
    document.body.classList.add('lock')
    document.body.style.paddingRight = scrollbarWidth + 'px';
    header.style.paddingRight = scrollbarWidth + 'px';

    const popup = document.getElementById('popup');
    popup.classList.add('active');
    popup.addEventListener('click', (e) => {
        if (!e.target.closest('.popup__content')) {
            closePopup();
        }
    })
}

function closePopup() {
    document.querySelector(".popup").classList.remove("active");
}

```

```

        setTimeout(() => {
            document.body.classList.remove('lock');
            document.body.style.paddingRight = '';
            document.querySelector('header').style.paddingRight = '';
            document.getElementById('association-field').value = '';
            document.getElementById('confirm-error').style.display =
'none';
        }, 300);
    }

function validateForm(form) {
    form.addEventListener("submit", function(e) {

        let email = document.getElementById("email").value;
        let password = document.getElementById("password").value;
        const err = document.querySelector(".registration-login-
form-error");
        const emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
        const passwordPattern = /^[A-Za-z0-9!@#$$%^&*()\-_+=.,<>/?\
[\]\{\}\|\`~]+$/;

        if (email.length < 4) {
            e.preventDefault();
            err.textContent = "Мінімальна довжина пошти - 4
СИМВОЛИ !";
        } else if (password.length < 4) {
            e.preventDefault();
            err.textContent = "Мінімальна довжина паролю - 4
СИМВОЛИ !";
        } else if (email.length > 50) {
            e.preventDefault();
            err.textContent = "Максимальна довжина пошти - 50
СИМВОЛІВ !";
        } else if (password.length > 50) {
            e.preventDefault();
            err.textContent = "Максимальна довжина паролю - 50
СИМВОЛІВ !";
        } else if (!emailPattern.test(email)) {
            e.preventDefault();
            err.textContent = "Невірний формат пошти !";
        } else if (!passwordPattern.test(password)) {
            e.preventDefault();
            err.textContent = "Пароль може складатись тільки з
латинських малих та великих літер, цифр та символів !@#$$%^&*()-
_+=.,<>/?[]{}\\|`~";
        }

    });
}

function floatUpLine(form) {
    const inputs = form.querySelectorAll('input');

```

```

inputs.forEach(input => {

    input.addEventListener('keydown', (e) => {
        if (e.key === ' ') {
            e.preventDefault();
        }
    });

    const placeholderText = input.getAttribute('placeholder')
|| '';
    input.removeAttribute('placeholder');

    const floatLabel = document.createElement('div');
    floatLabel.className = 'floating-placeholder';
    floatLabel.textContent = placeholderText;

    const wrapper = document.createElement('div');
    wrapper.className = 'form-group';
    input.parentNode.insertBefore(wrapper, input);
    wrapper.appendChild(input);
    wrapper.appendChild(floatLabel);

    input.addEventListener('focus', () => {
        floatLabel.classList.add('active-placeholder');
    });

    input.addEventListener('blur', () => {
        if (input.value.length === 0) {
            floatLabel.classList.remove('active-placeholder');
        }
    });

    if (input.value.length !== 0) {
        floatLabel.classList.add('active-placeholder');
    }
});

}

const rForm = document.getElementById("r-form");
const lForm = document.getElementById("l-form");
const forms = [lForm, rForm];

window.addEventListener("load", () => {
    forms.forEach(form => {
        if (form !== null) {
            validateForm(form);
            floatUpLine(form);
        }
    });
});

});

document.addEventListener('DOMContentLoaded', () => {

```

```

        setTimeout(() => {
            document.body.classList.add('transitions-enabled');
        }, 100);
    });

const gForm = document.getElementById('g-form');
if (gForm !== null) {
    document.querySelectorAll('.checkbox').forEach(checkbox => {
        const input =
checkbox.querySelector('input[type="checkbox"]');
        if (input.checked) {
            checkbox.classList.add('active');
        }
        checkbox.addEventListener('click', () => {
            if (checkbox.classList.contains('active')) {
                input.checked = false;
            } else {
                input.checked = true;
            }
            checkbox.classList.toggle('active');
        });
    });

    const rangeInput = document.querySelector('.length-range');
    let rangeValue = document.getElementById('range-value');
    rangeValue.innerText = `${rangeInput.value}`;
    rangeInput.addEventListener('input', () => {
        rangeValue.innerText = `${rangeInput.value}`;
    });
}

function setScrollToSession() {
    sessionStorage.setItem('scrollPos', window.scrollY);
}

const updateForm = document.querySelector('.popup__content');
if (updateForm) {
    const scrollPos = sessionStorage.getItem('scrollPos');
    if (scrollPos) {
        window.scrollTo(0, parseInt(scrollPos));
        sessionStorage.removeItem('scrollPos');
    }
    userInput = document.getElementById('association-field');
    confirmError = document.getElementById('confirm-error');
    updateForm.addEventListener('submit', (e) => {
        if (userInput.value.length == 0) {
            confirmError.style.display = 'block';
            confirmError.innerText = "Ви нічого не ввели !";
            e.preventDefault();
        } else if (userInput.value.length > 100) {
            confirmError.style.display = 'block';

```



```

        confirmPassword.innerText = "Довжина опису має НЕ
перевищувати 100 символів !";
        e.preventDefault();
    }
});
}

```

style.css

```

/* Скидаємо усі непотрібні стилі */
*{
    padding: 0;
    margin: 0;
    border: 0;
}
*, *:before, *:after{
    -moz-box-sizing: border-box;
    -webkit-box-sizing: border-box;
    box-sizing: border-box;
}
:focus, :active{outline: none;}
a:focus, a:active{outline: none;}
nav, footer, header, aside{display: block;}
html, body{
    height: 100%;
    width: 100%;
    font-style: 100%;
    line-height: 1;
    font-size: 14px;
    -ms-text-size-adjust: 100%;
    -moz-text-size-adjust: 100%;
    -webkit-text-size-adjust: 100%;
}
input, button, textarea{font-family: inherit;}
input::-ms-clear{display:none;}
button{cursor: pointer;}
button::-moz-focus-inner{padding: 0;border: 0;}
a, a:visited{text-decoration: none;}
a:hover{text-decoration: none;}
ul li{list-style-type: none;}
img{vertical-align: top;}
h1, h2, h3, h4, h5, h6{font-style: inherit; font-weight: 400;}

/* Основні стилі */

body {
    background-color: rgba(1, 52, 117, 255);
    height: 100vh;
    margin: 0;
    color: white;
    font-family: "Arial";
}

```

```

body.lock {
  overflow: hidden;
}

header {
  height: 100px;
  width: 100%;
  background-color: #e1e1e1;
  position: fixed;
  top: 0;
  left: 0;
  z-index: 5;
}

.header-content {
  color: black;
  max-width: 1080px;
  margin: 0 auto;
  padding: 0 20px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  height: 100%;
  font-size: 21px;
  font-weight: bold;
  @media (max-width: 1280px) {
    max-width: 760px;
    font-size: 19px;
  }
  @media (max-width: 900px) {
    max-width: 550px;
    font-size: 17px;
  }
}

.header-button {
  width: 200px;
  height: 55px;
  display: flex;
  justify-content: center;
  align-items: center;
  border: 5px solid rgba(0, 52, 172, 255);
  background-color: rgba(0, 52, 172, 0.3);
  border-radius: 15px;
  color: black;
  transition-duration: 0.4s;
  padding: 0 5px;
  margin: 0 5px;
}

.header-button:hover {
  background-color: rgba(0, 52, 172, 255);
}

```

```

        color: rgba(255, 255, 255, 0.85);
    }

    .reg-log-button-back {
        margin: 0 auto;
    }

    .container {
        color: white;
        min-height: 100vh;
        max-width: 1080px;
        margin: 0 auto;
        background-color: rgba(0, 0, 0, 0.9);
        padding: 20px;
        padding-top: 130px;
        text-align: justify;

        p, div, input, button {
            font-size: 21px;
            line-height: 1.3;
        }

        @media (max-width: 1280px) {
            max-width: 760px;
            p, div, input, button {
                font-size: 19px;
                line-height: 1.2;
            }
        }

        @media (max-width: 900px) {
            max-width: 550px;
            p, div, input, button {
                font-size: 17px;
                line-height: 1.1;
            }
        }
    }

    .registration-login-text {
        text-align: center;
    }

    .registration-login-form {
        max-width: 500px;
        margin: 30px auto 0 auto;
        button { width: 100%; }
    }

    .form-control {
        display: block;
        width: 100%;
        padding: 10px 60px 10px 10px;
    }

```

```

    margin-top: 50px;
    opacity: 0.8;
    border-radius: 5px;
}

.registration-login-button {
    margin-top: 30px;
    width: 230px;
    height: 55px;
    border: 5px solid rgba(0, 52, 172, 255);
    background-color: rgba(255, 255, 255, 0.8);
    border-radius: 15px;
    color: rgba(0,0,0,0.85);
    font-weight: bold;
    transition-duration: 0.4s;
}

.registration-login-button:hover {
    background-color: rgba(0, 52, 172, 255);
    color: rgba(255, 255, 255, 0.85);
}

.registration-login-form-error {
    margin-top: 30px;
    color: red;
    font-weight: bold;
    animation: blink 1.5s ease-in-out infinite;
    opacity: 0;
}

.password-container {
    position: relative;
}

.eye-icon {
    cursor: pointer;
    width: 35px;
    position: absolute;
    top: 6px;
    right: 10px;
    @media (max-width: 1280px) {
        width: 30px;
    }
}

.email-title {
    display: flex;
    align-items: center;
    img {
        margin-right: 10px;
        width: 30px;
    }
}

```

```

        @media (max-width: 900px) {
            p { display: none; }
        }
    }

    @keyframes blink {
        0%, 100% { opacity: 1; }
        50% { opacity: 0.1; }
    }

    .form-group {
        position: relative;
        margin-bottom: 20px;
    }

    .floating-placeholder {
        position: absolute;
        top: 10px;
        left: 10px;
        font-size: 16px;
        color: rgba(0, 0, 0, 0.8);
        pointer-events: none;
        transition: 0.3s ease-in-out all;
    }

    .active-placeholder {
        top: -33px;
        left: 5px;
        font-size: 11px;
        color: white;
        padding: 0 4px;
        @media (max-width: 900px) {
            top: -28px;
        }
    }

    .generation {
        max-width: 500px;
        margin: 0 auto;
        .form-control {
            margin: 20px 0;
        }
        .registration-login-button {
            width: 100%;
            margin: 20px 0;
        }
    }

    .generation-form__main {
        display: flex;
        flex-direction: column;
    }

```

```

}

.generation-form__settings {
  div {
    margin-top: 35px;
  }
  display: flex;
  flex-direction: column;
}

.checkbox {
  align-self: flex-start;
  position: relative;
  cursor: pointer;
  input {
    display: none;
  }
  padding-left: 70px;
}

.checkbox::before {
  position: absolute;
  left: 0;
  top: 3px;
  content: "";
  width: 50px;
  height: 20px;
  border-radius: 10px;
  background-color: rgba(255, 255, 255, 0.85);
  @media (max-width: 1280px) {
    top: 1px;
  }
  @media (max-width: 900px) {
    top: -1px;
  }
}

.checkbox::after {
  position: absolute;
  top: -1px;
  left: -2px;
  content: "";
  width: 28px;
  height: 28px;
  border-radius: 50%;
  background-color: rgba(255, 255, 255, 1);
  @media (max-width: 1280px) {
    top: -3px;
  }
  @media (max-width: 900px) {
    top: -5px;
  }
}

```

```

}

body.transitions-enabled .checkbox::before,
body.transitions-enabled .checkbox::after {
    transition: all 0.3s ease;
}

.checkbox.active::after {
    left: 30px;
}

.checkbox.active::before {
    background-color: rgba(0, 52, 172, 1);
}

.length-range {
    margin-top: 10px;
    width: 100%;
    cursor: pointer;
}

.input-password {
    position: relative;
}

.copy-btn {
    position: absolute;
    width: 35px;
    height: 35px;
    top: 26px;
    @media (max-width: 1280px) {
        width: 30px;
        height: 30px;
        top: 26px;
    }
    @media (max-width: 900px) {
        width: 28px;
        height: 28px;
        top: 25px;
    }
    right: 15px;
    cursor: pointer;
    background-image: url('/static/img/copy.png');
    background-size: contain;
    background-color: transparent;
    transition: all 0.2s ease;
}

.copy-btn:active {
    transition: all 0.2s ease;
    transform: scale(0.9);
    opacity: 0.8;
}

```

```

}

.history {
    max-width: 500px;
    margin: 0 auto;
    text-align: left;
}

.history-item {
    margin-top: 35px;
    .input-password input{
        margin-top: 10px;
        padding-right: 150px;
    }
    .copy-btn {
        right: 10px;
        top: 6px;
        @media (max-width: 900px) {
            top: 6px;
        }
    }
}

.history-item:first-child {
    margin-top: 0;
}

.history-row {
    word-wrap: break-word;
}

.delete-btn {
    display: block;
    width: 30px;
    height: 30px;
    background-image: url('/static/img/delete.png');
    background-size: contain;
    background-color: transparent;
    cursor: pointer;
    position: absolute;
    width: 40px;
    height: 40px;
    top: 4px;
    right: 60px;
    @media (max-width: 1280px) {
        width: 36px;
        height: 36px;
        top: 4px;
        right: 58px;
    }
    @media (max-width: 900px) {
        width: 32px;
    }
}

```



```

        height: 32px;
        top: 3px;
        right: 56px;
    }
}

.update-btn {
    display: block;
    width: 30px;
    height: 30px;
    background-image: url('/static/img/pencil.png');
    background-size: contain;
    background-color: transparent;
    cursor: pointer;
    position: absolute;
    width: 36px;
    height: 36px;
    top: 6px;
    right: 114px;
    @media (max-width: 1280px) {
        width: 31px;
        height: 31px;
        top: 6px;
        right: 110px;
    }
    @media (max-width: 900px) {
        width: 27px;
        height: 27px;
        top: 6px;
        right: 103px;
    }
}

.popup {
    position: fixed;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.8);
    top: 0;
    left: 0;
    opacity: 0;
    visibility: hidden;
    z-index: 10;
    overflow-y: auto;
    overflow-x: hidden;
    transition: all 0.5s ease 0s;
}

.popup__body {
    min-height: 100%;
    display: flex;
    align-items: center;

```

```

        justify-content: center;
        padding: 30px 10px;
    }

    .popup__body form {
        background-color: rgba(255, 255, 255, 255);
        color: #000;
        max-width: 300px;
        padding: 15px;
        text-align: left;
        transition: all 0.5s ease 0s;
        border-radius: 5px;
        opacity: 0;
        transform: perspective(600px) translate(0px, -150%)
        rotateX(45deg);
    }

    .popup.active {
        opacity: 1;
        visibility: visible;
        form {
            opacity: 1;
            transform: perspective(600px) translate(0px, 0%)
            rotateX(0deg);
        }
    }

    .popup__content {
        p {
            margin-bottom: 15px;
        }
        #association-field {
            width: 100%;
            border-bottom: 2px solid rgba(0, 0, 0, 0.6);
            padding: 4px 4px 4px 0;
        }
    }

    .confirm-buttons {
        margin-top: 23px;
        display: flex;
        flex-direction: row;
        justify-content: space-between;
        button {
            min-width: 100px;
            padding: 7px 5px;
            border-radius: 10px;
            border: 4px solid rgba(0, 52, 172, 255);
            background-color: rgba(0, 52, 172, 0.3);
            transition: all 0.4s ease 0s;
        }
        button:hover {

```

```
        background-color: rgba(0, 52, 172, 255);
        color: rgba(255, 255, 255, 0.85);
    }
}

#confirm-error {
    margin: 15px 0 0 0;
    color: red;
    font-weight: bold;
    animation: blink 1.5s ease-in-out infinite;
    opacity: 0;
    display: none;
}
```