

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кафедра програмної інженерії

КУРСОВА РОБОТА
ПОЯСНЮВАЛЬНА ЗАПИСКА
з дисципліни «Об'єктно-орієнтоване програмування»
Менеджер паролів

Керівник, проф.
Студент гр. ПЗП-22-6

Бондарєв В.М.
Горішня К.О.

Комісія:

Проф.	_____	Бондарєв В.М.
Ст. викл.	_____	Черепанова Ю. Ю.
Ст. викл.	_____	Ляпота В.М.

Харків 2023

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра: *програмної інженерії*Рівень вищої освіти: *перший (бакалаврський)*Дисципліна: *Об'єктно-орієнтоване програмування*Спеціальність: *121 Інженерія програмного забезпечення*Освітня програма: *Програмна інженерія*Курс 1Група ПЗПІ-22-6Семестр 2**ЗАВДАННЯ**
*на курсовий проект студента**Горішньої Катерини Олексіївни*

1 Тема проекту:

*Менеджер паролів*2 Термін здачі студентом закінченого проекту: ***“26” - травня - 2023 р.***

3 Вихідні дані до проекту:

Специфікація програми, методичні вказівки до виконання курсової роботи

4 Зміст розрахунково-пояснювальної записки:

Вступ, опис вимог, проектування програми, інструкція користувача, висновки

КАЛЕНДАРНИЙ ПЛАН

<i>№</i>	<i>Назва етапу</i>	<i>Термін виконання</i>
1	Видача теми, узгодження і затвердження теми	06.03.2023 р.
2	Формулювання вимог до програми	18.03.2023 – 27.03.2023 р.
3	Робота над архітектурою програми, створення плану роботи	05.04.2023 – 14.04.2023 р.
4	Розробка функцій реєстрації облікового запису	15.04.2023 – 22.04.2023 р.
5	Розробка функцій додавання, редагування та видалення паролів	23.04.2023 – 03.05.2023 р.
6	Розробка користувацького інтерфейсу	06.05.2023 – 12.05.2023 р.
7	Підключення симетричного шифрування рядків до програми	13.05.2023 – 18.05.2023 р.
8	Тестування і доопрацювання розробленої програмної системи	20.05.2023 – 22.05.2023 р.
9	Оформлення пояснювальної записки, додатків, графічного матеріалу	24.05.2023 – 26.05.2023 р.
10	Захист	05.06.2023 – 16.06.2023 р.

Студент _____

Керівник _____

Бондарєв В.М._____

(Прізвище, Ім'я, По батькові)

«13» лютого _____ 2022 р.

РЕФЕРАТ

Пояснювальна записка до курсової роботи: 63 с., 20 рис., 1 додаток, 5 джерел.

ВЕБ-ДОДАТОК, ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ПАРОЛІ, ЗАХИСТ, ANGULAR, JAVASCRIPT, БАЗИ ДАНИХ, СИМЕТРИЧНЕ ШИФРУВАННЯ.

Мета роботи – створити веб-додаток, який дозволить користувачам зручно та безпечно зберігати свої логіни та паролі від різних мереж. Такі сервіси стали дуже актуальними в останні роки, оскільки користувачі використовують все більше ресурсів, які вимагають введення паролів.

У результаті роботи над створенням веб-додатку було дотримано календарний план та отримано готовий проєкт під назвою «Менеджер паролів». Це важливий та актуальний інструмент, де користувачі мають можливість зберігати та керувати своїми паролями в безпечному та доступному місці. Він допоможе користувачам захистити свою конфіденційну інформацію, зменшить ризик втрати або забуття паролів. Завдяки його функціональності, користувачі можуть зручно і безпечно зберігати, додавати, редагувати та видаляти паролі, а також створювати облікові записи, щоб мати зручний доступ до паролів з будь-якого місця за умови наявності Інтернету.

В процесі розробки було використано середовище Webstorm, мова програмування JavaScript, фреймворк Angular, Node.js, фреймворк Express.js, база даних MongoDB, платформа для тестування API Postman, бібліотека для симетричного шифрування рядків CryptoJS.

ЗМІСТ

Вступ.....	6
1 Опис вимог.....	7
1.1 Функціонал програми.....	7
1.1.1 Функція «Реєстрація облікового запису».....	8
1.1.2 Функція «Вхід в обліковий запис».....	9
1.1.3 Функція «Вихід з облікового запису».....	11
1.1.4 Функція «Додавання логіна/пароля».....	12
1.1.5 Функція «Редагування логіна/пароля».....	13
1.1.6 Функція «Показ пароля».....	14
1.1.7 Функція «Видалення логіна/пароля».....	15
1.2 Інтерфейс програми.....	16
2 Проектування програми.....	18
3 Інструкція користувача.....	22
3.1 Реєстрація/вхід до свого облікового запису.....	22
3.2 Робота з додаванням логіна/пароля.....	26
3.3 Робота з редагуванням логіна/пароля.....	28
3.4 Робота з відображенням пароля.....	30
3.5 Робота з видаленням логіна/пароля.....	31
3.6 Вихід зі свого облікового запису.....	32
Висновки.....	33
Перелік джерел посилання.....	34
Додаток А Код програми.....	35

ВСТУП

В даний час зростаюча кількість онлайн-сервісів, що вимагають реєстрації та входу за допомогою логінів та паролів, створює складність для користувачів у керуванні своїми обліковими даними. Часто виникає потреба у збереженні багатьох паролів для різних сервісів, що може призвести до забуття або використання ненадійних методів збереження паролів.

Мета роботи – створити веб-додаток, який дозволить користувачам зручно та безпечно зберігати свої логіни та паролі від різних мереж. Наш веб-додаток буде надавати користувачам зручний спосіб керування паролями, де вони зможуть легко додавати нові паролі, змінювати, видаляти та переглядати їх у безпечному середовищі.

Однією з головних переваг веб-додатку «Менеджер паролів» є захист конфіденційної інформації. Користувачі, які застосовують один і той же пароль для різних вебсайтів, ризикують, що їхні дані можуть бути скомпрометовані у випадку, якщо один із сайтів буде зламаний. «Менеджер паролів» дозволяє використовувати унікальні та складні паролі для кожного вебсайту, що зменшує ризик крадіжки їхніх даних.

Іншою важливою функцією веб-додатку є легкий доступ до паролів з будь-якого пристрою з доступом до Інтернету. Користувачам більше не потрібно запам'ятовувати або записувати свої паролі на папері, щоб мати до них доступ. Можна просто зайти у свій обліковий запис зі свого комп'ютера, планшета або смартфона та знайти необхідний пароль.

Після впровадження веб-додатку, типова діяльність користувача значно спроститься та збільшиться рівень безпеки їх паролів. Замість того, щоб запам'ятовувати і вводити паролі вручну, користувачі мають доступ до централізованого інтерфейсу і можуть легко керувати обліковими даними.

1 ОПИС ВИМОГ

1.1 Функціонал програми

Проаналізувавши технічне завдання та оцінивши мої можливості, враховуючи відведений для роботи час, було розроблено вимоги до веб-додатку та складено план робіт над ним. Веб-додаток буде розроблений з використанням сучасних технологій та практик. Вхідні дані, які користувач буде вводити на фронтенді, будуть зашифровані за допомогою бібліотеки CryptoJS [1], забезпечуючи конфіденційність та безпеку. Зашифровані дані будуть передані на бекенд, де вони будуть оброблені та збережені.

Основною структурою веб-додатку буде Single-Page Application (SPA) [2], що дозволить створити ефективний інтерфейс користувача. Це означає, що при зміні даних на одній сторінці, інші сторінки будуть автоматично оновлюватись, відображаючи актуальну інформацію. Такий підхід значно полегшує навігацію користувача та забезпечує швидку відповідь системи на його дії. Окрім того, для забезпечення безпеки та ідентифікації користувачів, використовується JWT (JSON Web Token) [3]. Кожен користувач отримує унікальний JWT після успішного входу в свій обліковий запис, який потім використовується для перевірки прав доступу та автоматичної ідентифікації при подальших запитах.

Веб-додаток надасть користувачу наступні можливості:

- можливість створити обліковий запис;
- можливість увійти в обліковий запис;
- можливість вийти з облікового запису;
- можливість додавати логіни/паролі;
- можливість редагувати логіни/паролі;
- можливість видаляти логіни/паролі.

1.1.1 Функція «Реєстрація облікового запису»

Ця функція дозволяє користувачеві створити новий обліковий запис у веб-додатку. Користувач буде мати можливість ввести свої особисті дані (ім'я, пошту) та створити пароль для входу в систему.

При запуску веб-додатку на екрані з'являється сторінка реєстрації «SIGN UP» (рис. 1.1), яка містить наступні поля введення тексту:

- Name (ім'я): користувач вводить своє ім'я;
- Email (електронна пошта): користувач вводить свою дійсну електронну адресу;
- Password (пароль): користувач вводить пароль для облікового запису.

Користувач заповнює необхідні поля і натискає кнопку «SIGN UP» для створення облікового запису. При реєстрації веб-додаток використовує метод перевірки правильності введених даних, такі як валідація електронної пошти.

Якщо дані коректні, обліковий запис успішно створюється.

Якщо введені дані некоректні або виникає помилка, користувач отримує відповідне повідомлення про помилку і має можливість виправити неправильно введені дані.

Після успішної реєстрації користувач може використовувати свої облікові дані для входу до свого особистого кабінету.

Якщо користувач вже мав обліковий запис, то він натискає «Do you already have an account?» та переходить на сторінку «SIGN IN» (увійти до свого облікового запису).

SIGN UP

Name

Email

Password

SIGN UP

Do you already have an account?

Рисунок 1.1 – Ескіз користувацького інтерфейсу сторінки «SIGN UP»

1.1.2 Функція «Вхід в обліковий запис»

Ця функція дозволяє зареєстрованим користувачам виконати вхід в свій обліковий запис у веб-додатку. Користувач повинен ввести свою пошту та пароль, щоб отримати доступ до функціоналу веб-додатку.

Користувач заходить у веб-додаток, натискає, що він вже має обліковий запис, і перед ним з'являється сторінка «SIGN IN» (рис. 1.2), яка містить наступні поля введення тексту:

- Email (електронна пошта): користувач вводить пошту, на яку був зареєстрований обліковий запис.

– Password (пароль): користувач вводить свій пароль, пов'язаний з обліковим записом.

Користувач заповнює поля вірними обліковими даними і натискає кнопку «SIGN IN».

Система перевіряє введені дані для забезпечення відповідності пошти та пароля зареєстрованим обліковим даним.

Якщо введені дані є коректними, користувач успішно входить в свій обліковий запис і отримує повний доступ до функціоналу веб-додатку.

Якщо введені дані некоректні або не співпадають з зареєстрованими обліковими даними, користувач отримує відповідне повідомлення про помилку і може виправити введені дані.

A hand-drawn sketch of a login form titled "SIGN IN". The form is enclosed in a rectangular border. Inside the border, the text "SIGN IN" is written at the top. Below it, there are two input fields: the first is labeled "Email" and the second is labeled "type here password". Below the password field, there is a button labeled "SIGN IN". At the bottom of the form, there is a link labeled "Create new account".

Рисунок 1.2 – Ескіз користувацького інтерфейсу сторінки «SIGN IN»

1.1.3 Функція «Вихід з облікового запису»

Ця функція дозволяє зареєстрованим користувачам вийти зі свого облікового запису. Після виходу користувач втрачає доступ до функціоналу веб-додатку, який був доступний йому в рамках його облікового запису.

Користувач знаходиться у веб-додатку на головній сторінці «DASHBOARD» (рис. 1.3). Йому потрібно натиснути на опцію «LOGOUT». Після система завершує сеанс облікового запису, видаляє всі дані про користувача з поточної сесії і переключає користувача до сторінки «SIGN UP». У разі потреби користувач може знову увійти в свій обліковий запис.

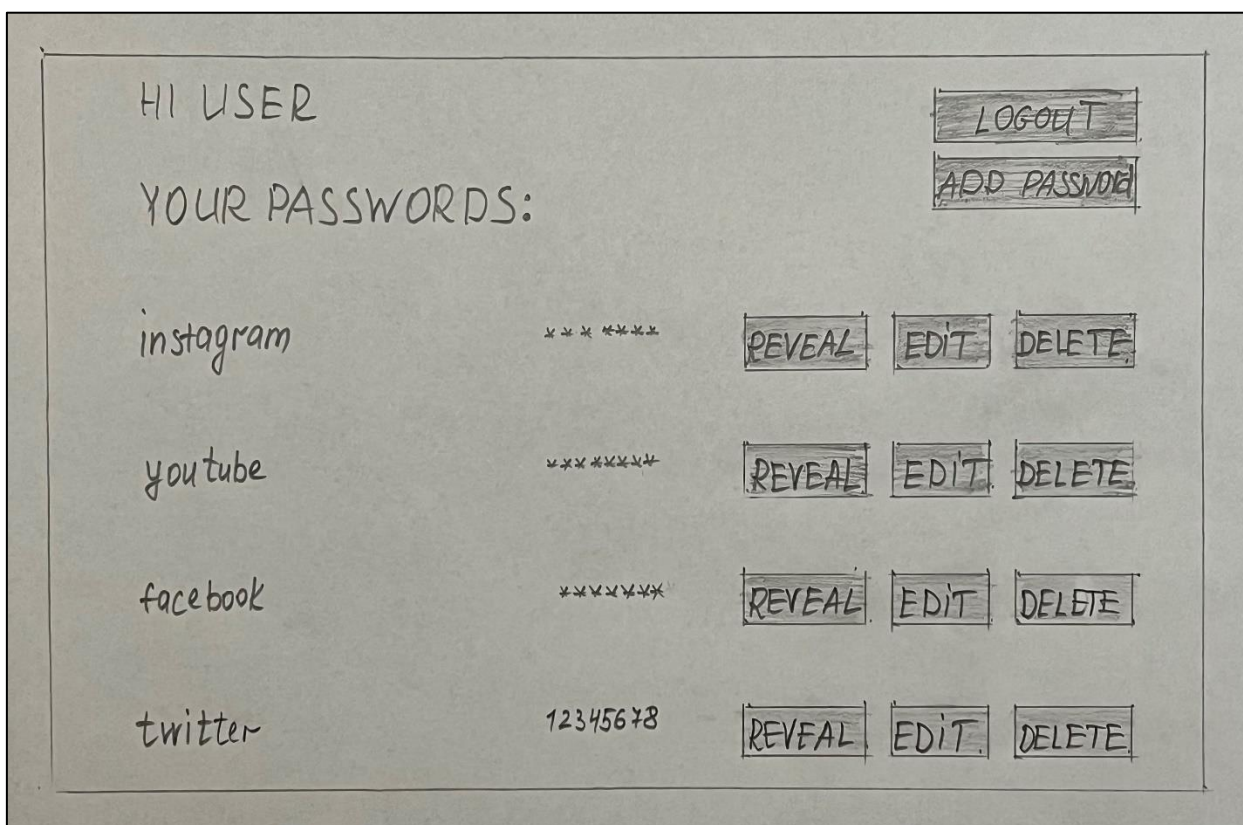


Рисунок 1.3 – Ескіз користувацького інтерфейсу сторінки «DASHBOARD»

1.1.4 Функція «Додавання логіна/пароля»

Ця функція дозволяє користувачеві додавати нові записи логінів та паролів у веб-додатку. Користувач може використовувати цю функцію для збереження та організації своїх різних облікових даних, таких як логіни та паролі для веб-сайтів, додатків, соцмереж тощо.

Користувач знаходиться у веб-додатку на головній сторінці «DASHBOARD» (рис. 1.3), де відображається список його логінів та паролів.

Користувач натискає на кнопку «ADD PASSWORD», щоб розпочати процес додавання нового запису логіна/пароля.

З'являється сторінка «ADD PASSWORD» (рис. 1.4), яка містить поля для введення необхідної інформації.

Користувач заповнює поля сторінки необхідними даними, такими як:

- Title (назва/позначення запису): короткий опис або назва облікового запису для подальшого розпізнавання;
- Login (логін): ідентифікатор або ім'я, яке використовується для входу в зазначений ним обліковий запис.
- Password (пароль): секретний рядок символів, що використовується для аутентифікації користувача в зазначений ним обліковий запис.

Після заповнення необхідних полів користувач натискає кнопку «CREATE» для створення нового запису логіна/пароля.

Система перевіряє, чи були введені необхідні дані. Якщо всі введені дані є коректними, новий запис логіна/пароля додається до бази даних.

Запис з'являється у списку логінів та паролів користувача, доступних для подальшого перегляду, редагування та використання.

Якщо користувач передумав створювати новий запис логіна/пароля, то він натискає «CANCEL» та повертається назад до «DASHBOARD».



CREATE A NEW PASSWORD

Title

Account Login

Password

CANCEL CREATE

Рисунок 1.4 – Ескіз користувацького інтерфейсу сторінки
«ADD PASSWORD»

1.1.5 Функція «Редагування логіна/пароля»

Ця функція дозволяє користувачеві зручно та ефективно вносити зміни до запису логіна та пароля. Вона допомагає підтримувати актуальність та точність облікових записів, забезпечуючи користувачеві оновлення існуючих даних та контроль над своїми логінами та паролями.

Користувач знаходиться у веб-додатку на головній сторінці «DASHBOARD» (рис. 1.3), де знаходиться список логінів та паролів, та знаходить бажаний запис логіна, який потребує редагування.

Користувач вибирає опцію «EDIT» для вибраного запису. Відкривається сторінка редагування «EDIT» (рис. 1.5), де можна змінювати поля, пов'язані з

ім'ям, логіном та паролем. Користувач вносить необхідні зміни до полів.

Після завершення редагування користувач зберігає внесені зміни, натискаючи кнопку «SAVE». Зміни вступають в дію, і дані про ім'я, логін та пароль оновлюються згідно з внесеними змінами.

Якщо користувач передумав редагувати логін та пароль, він натискає «CANCEL» та повертається назад до «DASHBOARD».

Рисунок 1.5 – Ескіз користувацького інтерфейсу сторінки «EDIT»

1.1.6 Функція «Показ пароля»

Ця функція дозволяє користувачеві переглядати схований пароль, пов'язаний з конкретним записом логіна у веб-додатку. Це додатковий захист,

який дозволяє забезпечити конфіденційність користувачів та запобігти випадковому розкриттю пароля.

Користувач знаходиться у веб-додатку на головній сторінці «DASHBOARD» (рис. 1.3), де знаходиться список логінів та паролів, та обирає бажаний запис логіна, для якого потрібно переглянути пароль.

Користувач вибирає опцію «REVEAL» (показати пароль) для вибраного запису. Пароль, пов'язаний з обраним записом логіна, відображається користувачеві.

Після закінчення перегляду паролю користувач може закрити відображення паролю, натиснувши кнопку «HIDE».

1.1.7 Функція «Видалення логіна/пароля»

Ця функція дозволяє користувачеві видаляти непотрібні записи логінів та паролів з веб-додатку. Вона забезпечує можливість очищення бази даних від зайвих або застарілих облікових записів, сприяючи збереженню чистоти та організації облікових записів, що дозволяє користувачеві підтримувати актуальну та зручну колекцію логінів та паролів.

Користувач знаходиться у веб-додатку на головній сторінці «DASHBOARD» (рис. 1.3), де відображається список його логінів та паролів, та обирає бажаний запис логіна, який потрібно видалити.

Користувач обирає опцію «DELETE» для вибраного запису. Відкривається сторінка для видалення паролів «DELETE» (рис. 1.6), де користувач підтверджує видалення натиснувши кнопку «DELETE». Далі відбувається фактичне видалення запису логіна/пароля з бази даних.

Якщо користувач передумав видаляти запис, він натискає «CANCEL» та повертається назад до «DASHBOARD».

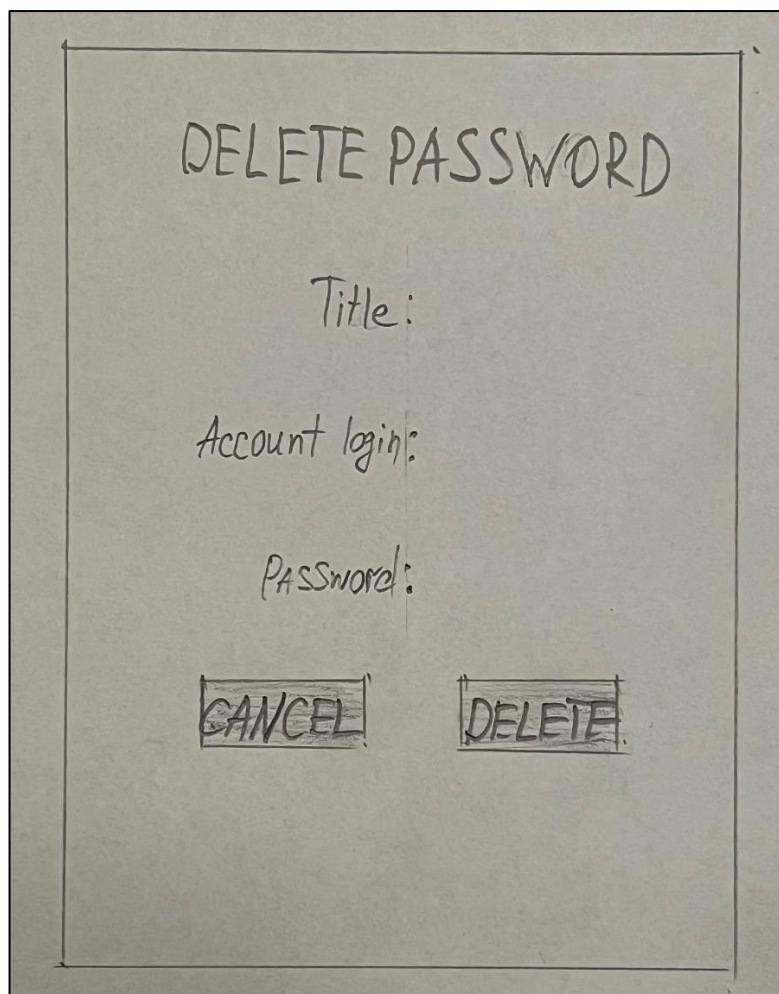


Рисунок 1.6 – Ескіз користувацького інтерфейсу сторінки «DELETE»

1.2 Інтерфейс програми

Наш веб-додаток має інтерфейс, що складається з шести сторінок, проте використовується підхід Single-Page Application (SPA). Це означає, що замість переходу зі сторінки на сторінку кожного разу, ми застосовуємо динамічне оновлення та підвантаження контенту на одній сторінці. Такий підхід дозволяє забезпечити більш плавну та швидку роботу програми.

Наш інтерфейс дозволяє користувачам додавати, видаляти та редагувати паролі, і зміни, які вони вносять, одразу ж відображаються на їх

«DASHBOARD», без необхідності перезавантаження сторінки. Це зберігає час та забезпечує зручну роботу з програмою. Враховуючи ці функціональності, наш веб-додаток надає зручну та безпечну роботу з управлінням паролями.

Сторінка «SIGN UP» (реєстрація). Ця сторінка дозволяє новим користувачам створити обліковий запис, ввести необхідну інформацію (ім'я, пошта та пароль) і створити кабінет для подальшого використання (рис. 1.1).

Сторінка «SIGN IN» (вхід до облікового запису). На цій сторінці користувачі можуть ввести свої облікові дані (пошту та пароль), щоб увійти до свого облікового запису і отримати доступ до збережених паролів (рис. 1.2).

Сторінка «DASHBOARD» (головна сторінка веб-додатку). Ця сторінка є центральним місцем управління паролями (рис. 1.3). Тут користувачі можуть переглядати список своїх збережених паролів, скористатися функціями додати пароль («ADD PASSWORD»), редагувати пароль («EDIT»), показати пароль («REVEAL»), видалити («DELETE») пароль, а також вийти зі свого облікового запису («LOGOUT»).

Сторінка «ADD PASSWORD» (додавання пароля). На цій сторінці користувачі можуть додавати нові паролі до свого облікового запису (рис. 1.4). Вони вводять необхідну інформацію, таку як назву, логін та пароль.

Сторінка «EDIT» (редагування пароля). На цій сторінці користувачі можуть вносити зміни до своїх наявних паролів (рис. 1.5). Вони можуть оновити поточні назви, логіни та паролі.

Сторінка «DELETE» (видалення пароля). Ця сторінка дозволяє користувачам видаляти непотрібні паролі зі свого облікового запису (рис. 1.6). Вони можуть вибрати пароль, який потрібно видалити, і пароль буде вилучено.

Кожна з цих сторінок має естетичний, простий та інтуїтивно зрозумілий дизайн, що дозволяє користувачам з легкістю виконувати необхідні дії та керувати своїми паролями з комфортом та безпекою.

2 ПРОЕКТУВАННЯ ПРОГРАМИ

Наш веб-додаток складається з двох основних частин: фронтенду і бекенду, які співпрацюють між собою для забезпечення функціональності і зручного користувацького досвіду (рис. 2.1).

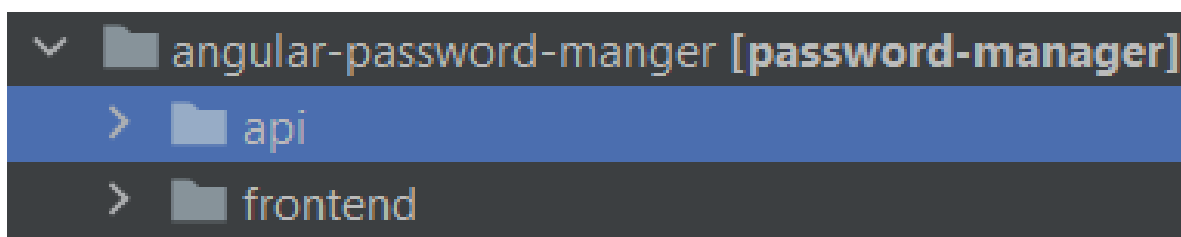


Рисунок 2.1 – Структура файлів веб-додатку «Менеджер паролів»

Бекенд веб-додатку складається з наступних компонентів (рис. 2.2).

База даних – для збереження різних файлів моделей, які визначають структуру і властивості даних.

Файл апі (файл app.js) – для налаштування серверу і виконання різних операцій, пов'язаних з обробкою запитів і відповідей.

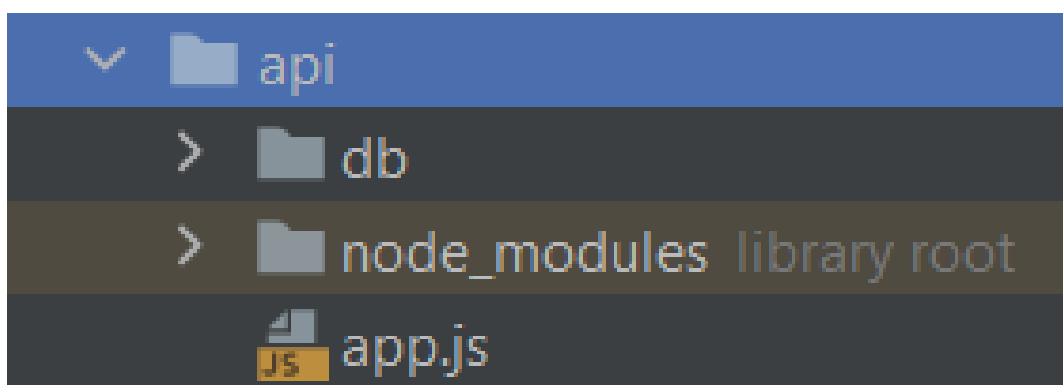


Рисунок 2.2 – Структура файлів бекендової частини веб-додатку

Конфігураційний файл Mongoose: ми використовуємо mongoose.js для налаштування з'єднання з базою даних, де є різні файли моделей (рис. 2.3).

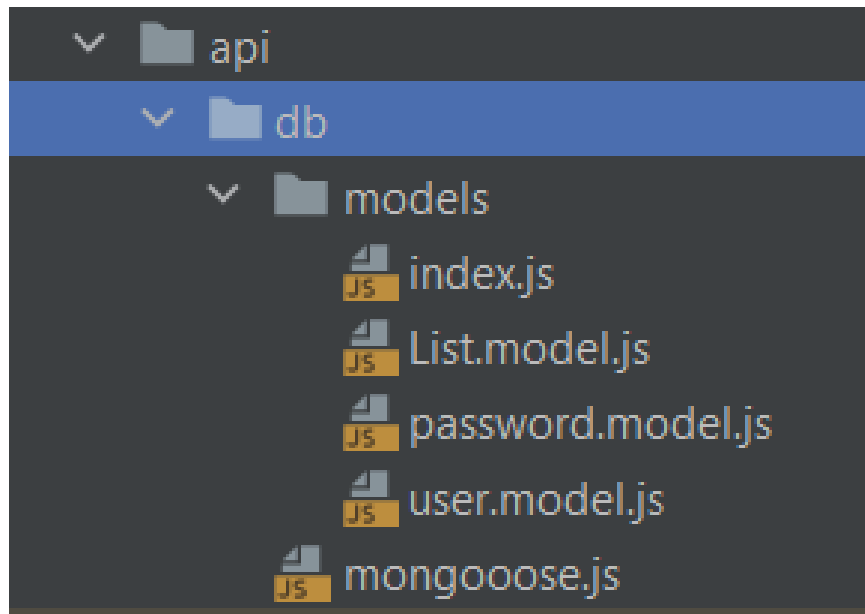


Рисунок 2.3 – Структура файлів для конфігурації бази даних

Фронтенд веб-додатку має наступну структуру (рис. 2.4).

Головний модуль (app.module) – для збірки всього фронтенду і встановлення основних налаштувань.

Різні сервіси, які містять методи для повторного використання і виконання різних завдань у програмі.

Інтерсептори – для перехоплення і обробки певних HTTP-запитів або відповідей до сервера [4].

Модуль маршрутизації – для забезпечення навігації по різних сторінках і визначення відповідних компонентів для кожного маршруту.

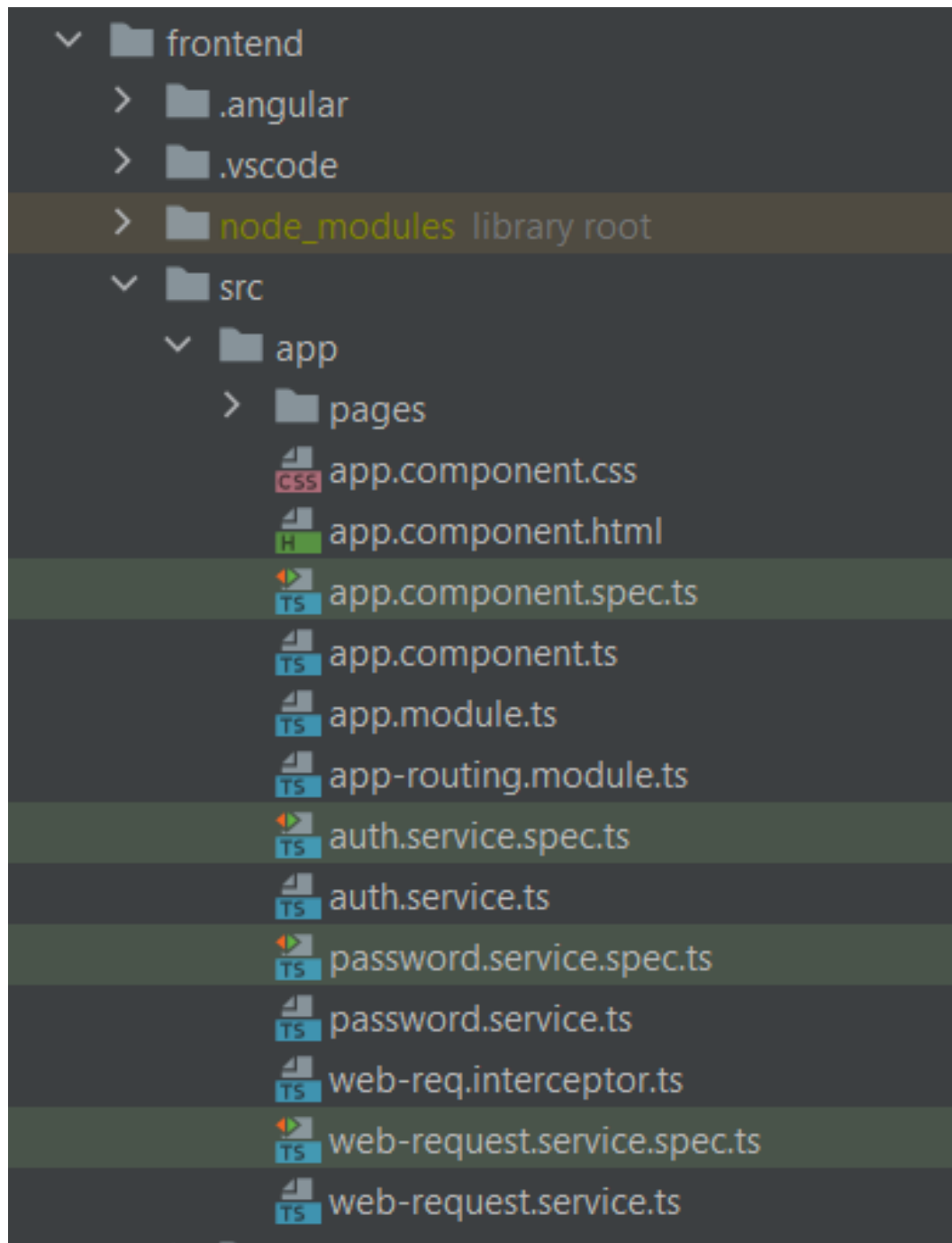


Рисунок 2.4 – Структура файлі фронтвої частини веб-додатку

Основною частиною нашого фронтенду є компоненти, які відповідають за відображення конкретних сторінок. Кожен компонент має свій TypeScript-файл для опису логіки, HTML-файл для розмітки та CSS-файл для стилізації [5]. Також ми використовуємо файл `spec.ts` для написання юніт-тестів, які перевіряють правильність роботи компонента (рис. 2.4).

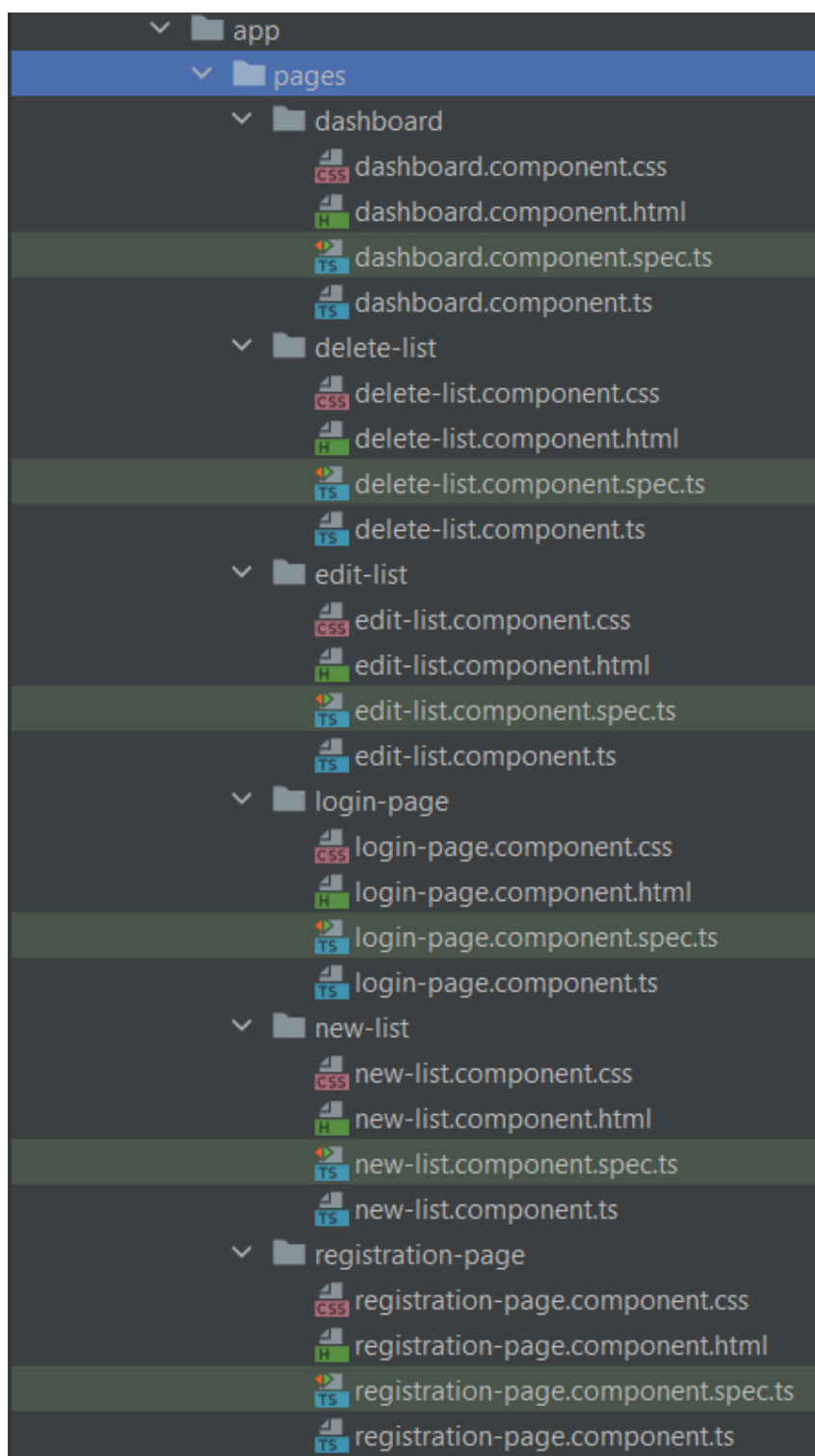


Рисунок 2.5 – Структура компонентів на фронтоній частині веб-додатку

Ця архітектура дозволяє нам ефективно розробляти та підтримувати наш веб-додаток, забезпечуючи розділення відповідальності між фронтендом і бекендом. Компоненти фронтенду дозволяють нам створювати динамічний інтерфейс, а бекенд забезпечує потрібні функції та зберігає дані в базі даних.

3 ІНСТРУКЦІЯ КОРИСТУВАЧА

В цьому розділі описана інструкція для користувача. Вона допоможе новачку розібратися в необхідних функціях програми та отримати бажаний результат від роботи з веб-додатком.

Наша програма є веб-додатком, а це означає, що Вам не потрібно завантажувати жодні файли. Для використання програми достатньо відкрити будь-який веб-браузер на будь-якому пристрої і знайти наш веб-додаток. Після цього Ви зможете зареєструватися або увійти до свого облікового запису, якщо він вже створений. Цей підхід дозволяє Вам зручно користуватися програмою без необхідності завантаження та встановлення додаткових файлів. Завдяки веб-інтерфейсу Ви можете отримати доступ до своїх даних з будь-якого місця та пристрою за умови наявності Інтернету.

3.1 Реєстрація/вхід до свого облікового запису

При запуску веб-додатку на екрані з'являється сторінка реєстрації «SIGN UP» (рис. 3.1), яка містить наступні поля введення тексту:

- Name (ім'я): користувач вводить своє ім'я;
- Email (електронна пошта): користувач вводить свою дійсну електронну адресу;
- Password (пароль): користувач вводить пароль для облікового запису.

Якщо у Вас немає облікового запису, то Ви маєте можливість ввести свої особисті дані (ім'я, пошту) та створити пароль для входу до системи.

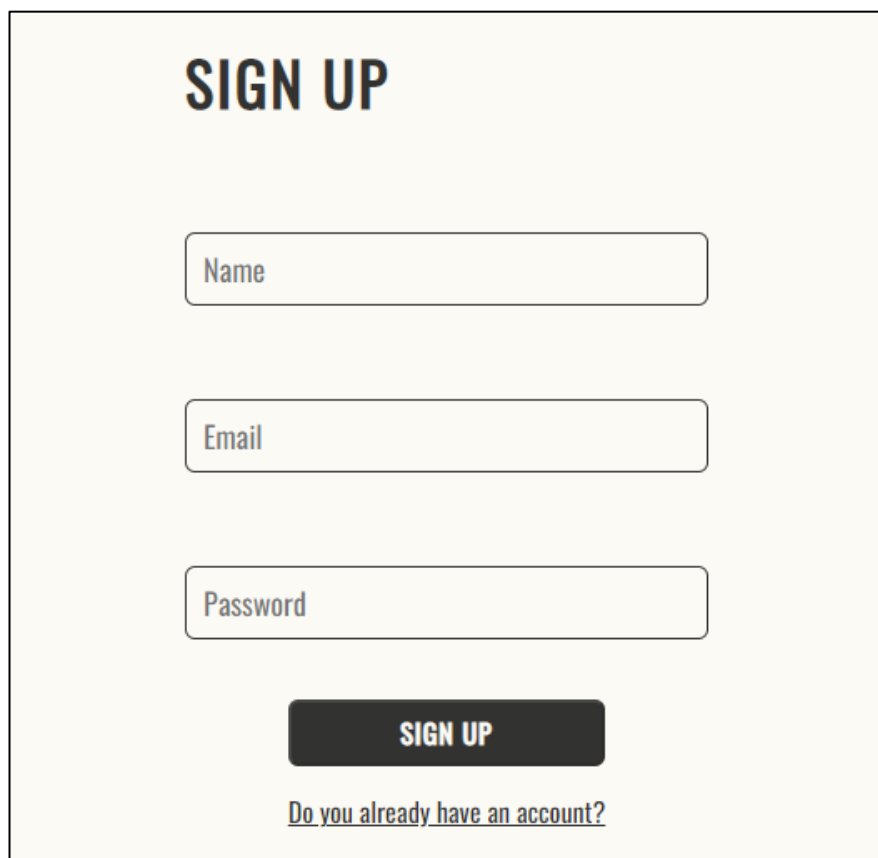
The image shows a 'SIGN UP' form interface. At the top, the text 'SIGN UP' is displayed in a large, bold, black font. Below this, there are three input fields: 'Name', 'Email', and 'Password', each with its label inside the field. The fields are rectangular with rounded corners and a thin border. Below the input fields is a dark grey button with the text 'SIGN UP' in white, bold, uppercase letters. At the bottom of the form, there is a link that reads 'Do you already have an account?' in a smaller, blue, underlined font.

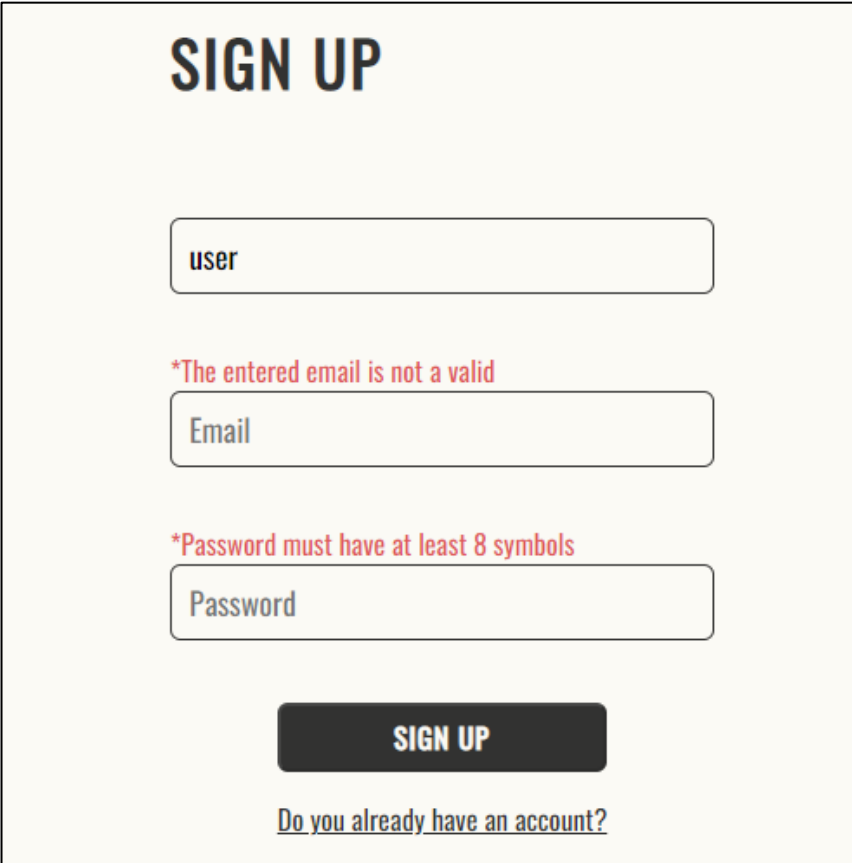
Рисунок 3.1 – Інтерфейс сторінки «SIGN UP»

Алгоритм реєстрації облікового запису проходить наступним чином.

Ви заповнюєте необхідні поля і натискаєте «SIGN UP» для створення свого облікового запису. При реєстрації використовується метод перевірки правильності введених даних, такий як валідація електронної пошти.

Якщо дані коректні, обліковий запис успішно створюється.

Якщо введені дані некоректні або виникає помилка, Ви отримуєте відповідне повідомлення про помилку і маєте можливість виправити неправильно введені дані (рис. 3.2).



The image shows a web form titled "SIGN UP" on a light beige background. It contains three input fields: a "user" field with the text "user" inside, an "Email" field with a red error message above it stating "*The entered email is not a valid", and a "Password" field with a red error message above it stating "*Password must have at least 8 symbols". Below the fields is a dark grey "SIGN UP" button. At the bottom, there is a blue hyperlink that reads "Do you already have an account?".

Рисунок 3.2 – Інтерфейс «SIGN UP» при некоректних введених даних

Після успішної реєстрації Ви можете використовувати свої облікові дані для входу до свого особистого кабінету.

Якщо Ви мали обліковий запис, то натискаєте «Do you already have an account?» та переходите на сторінку «SIGN IN» (увійти до облікового запису).

Алгоритм входу до облікового запису проходить наступним чином.

Перед Вами з’являється сторінка «SIGN IN» (рис. 3.3), яка містить наступні поля введення тексту:

- Email (електронна пошта): користувач вводить пошту, на яку був зареєстрований обліковий запис.

- Password (пароль): користувач вводить свій пароль, пов’язаний з обліковим записом.

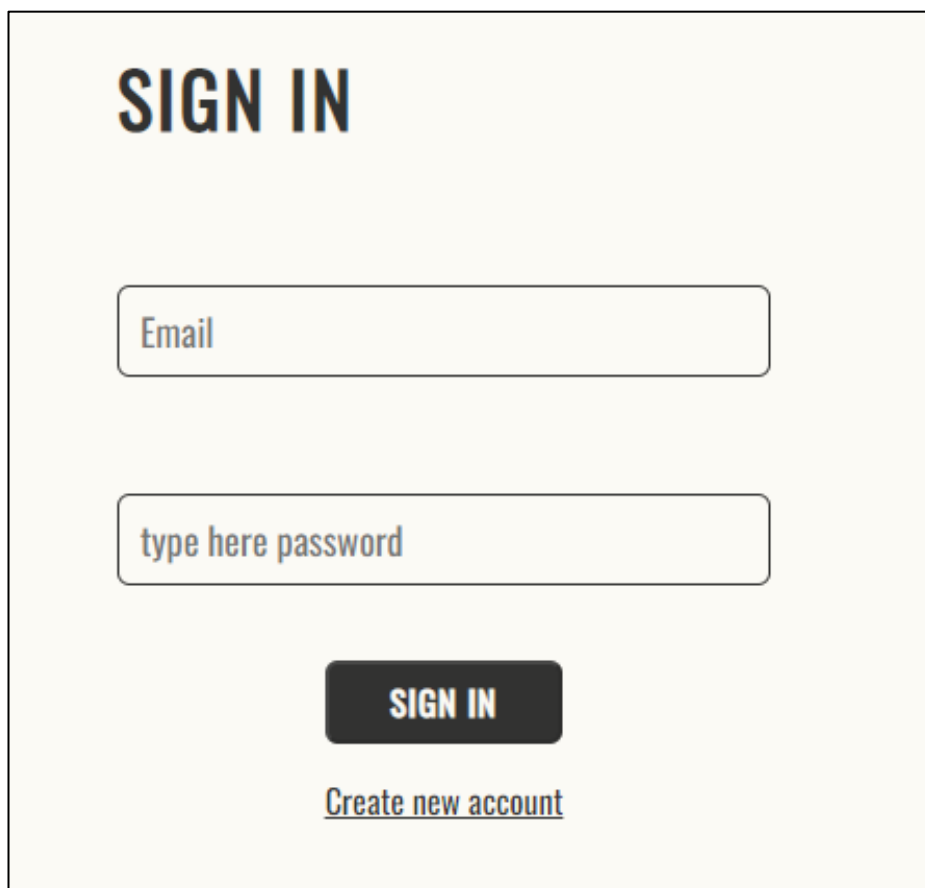
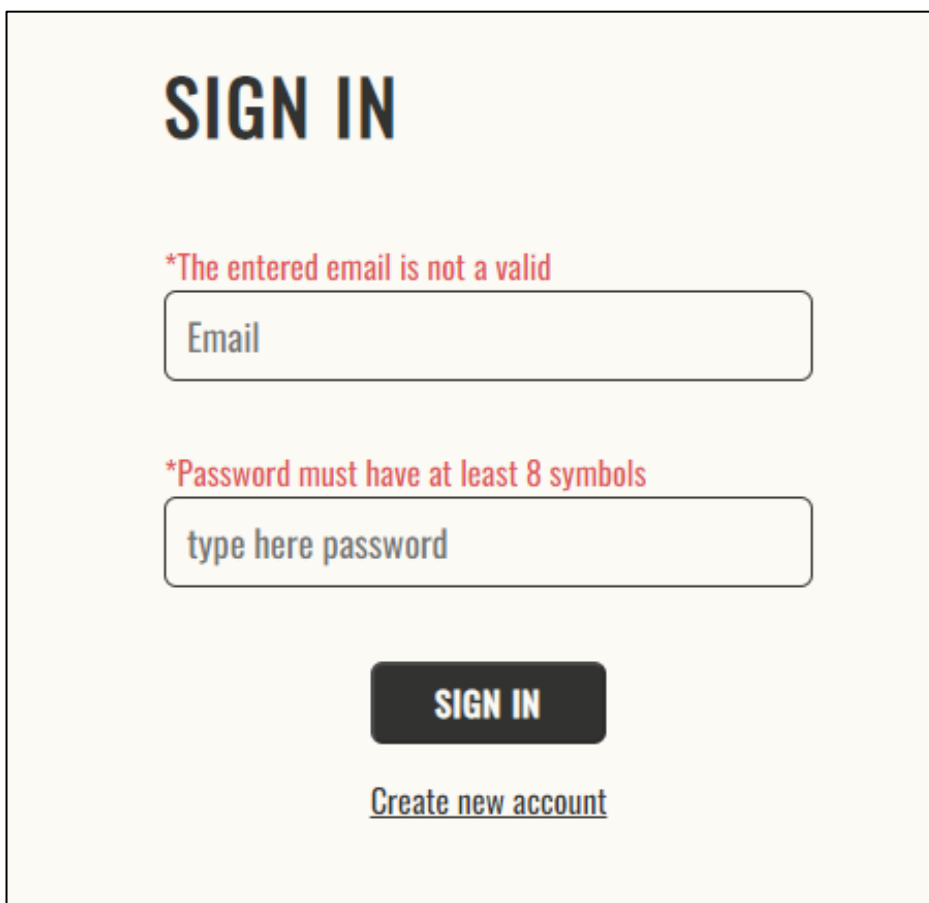
A sign-in form with a light beige background. At the top, the text "SIGN IN" is displayed in a large, bold, dark blue font. Below this, there are two input fields: the first is labeled "Email" in a small blue font, and the second is labeled "type here password" in a small blue font. Both fields have a thin grey border. Below the password field is a dark grey button with the text "SIGN IN" in a bold, white, sans-serif font. At the bottom of the form, there is a link that says "Create new account" in a blue, underlined font.

Рисунок 3.3 – Інтерфейс сторінки «SIGN IN»

Ви заповнюєте поля вірними обліковими даними і натискає кнопку «SIGN IN». Далі система перевіряє введені дані для забезпечення відповідності пошти та пароля зареєстрованим обліковим даним.

Якщо введені дані є коректними, користувач успішно входить в свій обліковий запис і отримує повний доступ до функціоналу програми.

Якщо введені дані некоректні або не співпадають з зареєстрованими обліковими даними, користувач отримує відповідне повідомлення про помилку і може виправити введені дані (рис. 3.4).



The image shows a 'SIGN IN' form with a light beige background. At the top, the text 'SIGN IN' is displayed in a large, bold, dark blue font. Below this, there are two error messages in red text: '*The entered email is not a valid' and '*Password must have at least 8 symbols'. The first error message is positioned above a white input field with a thin grey border, which contains the placeholder text 'Email'. The second error message is positioned above another white input field with a thin grey border, which contains the placeholder text 'type here password'. Below the password field, there is a dark grey button with the text 'SIGN IN' in white, bold, uppercase letters. At the bottom of the form, there is a link that says 'Create new account' in a blue, underlined font.

Рисунок 3.4 – Інтерфейс «SIGN IN» при некоректних введених даних

3.2 Робота з додаванням логіна/пароля

Після успішної реєстрації та входу до свого облікового запису, Ви потрапляєте до головної сторінки «DASHBOARD» (рис. 3.5), де знаходиться вже створений Вами список логінів та паролів.

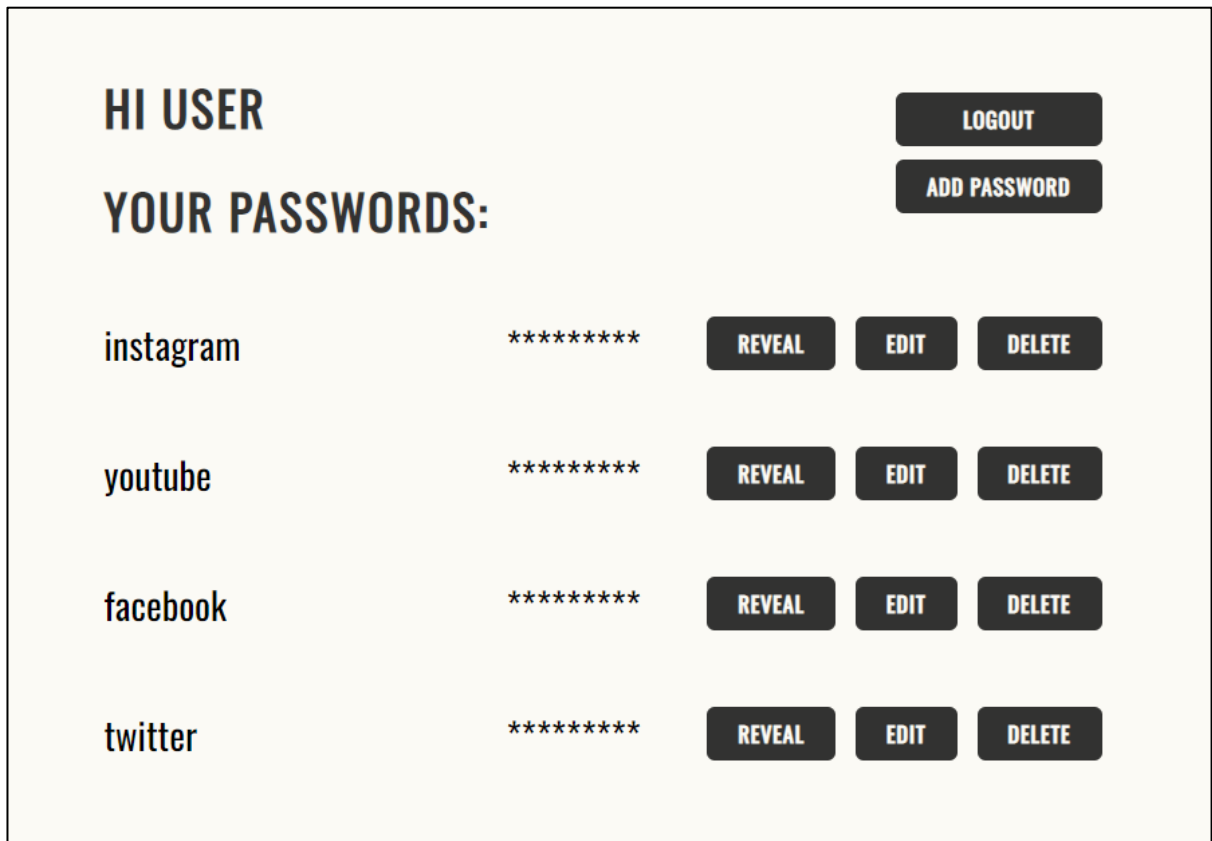


Рисунок 3.5 – Інтерфейс головної сторінки «DASHBOARD»

Ви отримуєте доступ до функції «Додавання логіна/пароля», що дозволяє додавати нові записи логінів та паролів у веб-додатку.

Алгоритм додавання нового логіну/пароля проходить наступним чином.

Ви натискаєте на кнопку «ADD PASSWORD», щоб розпочати процес додавання нового логіна/пароля. Перед Вами з'являється сторінка «ADD PASSWORD» (рис. 3.6), яка містить поля для введення необхідної інформації.

Ви заповнюєте поля сторінки необхідними даними, такими як:

- Title (назва/позначення запису): короткий опис або назва облікового запису для подальшого розпізнавання;
- Login (логін): ідентифікатор або ім'я, яке використовується для входу в зазначений ним обліковий запис.
- Password (пароль): секретний рядок символів, що використовується для аутентифікації користувачем в зазначений ним обліковий запис.



CREATE A NEW PASSWORD

*The title shouldn't be empty

Title

Account login (optional)

*The password shouldn't be empty

Password

CANCEL **CREATE**

Рисунок 3.6 – Інтерфейс сторінки «CREATE»

Після заповнення необхідних полів натискаєте кнопку «CREATE» для створення нового запису логіна/пароля.

Система перевіряє, чи були введені необхідні дані. Якщо всі введені дані є коректними, новий запис логіна/пароля додається до бази даних.

Запис з'являється у Вашому списку логінів та паролів, доступних для подальшого перегляду, редагування та використання.

Якщо Ви передумали створювати новий запис логіна/пароля, то Ви натискає «CANCEL» та повертаєтесь назад до сторінки «DASHBOARD».

3.3 Робота з редагуванням логіна/пароля

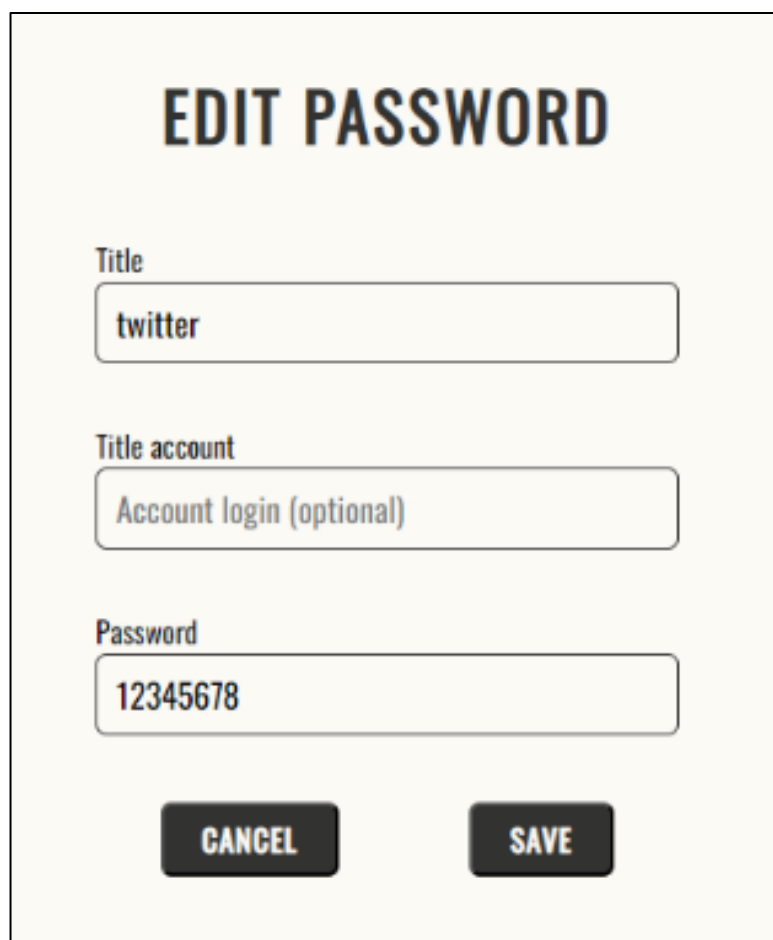
Після успішної реєстрації та входу в свій обліковий запис, Ви побачите головну сторінку «DASHBOARD» (рис. 3.5), де знаходяться вже створені

Вами логіни та паролі.

Ви отримуєте доступ до функції «Редагування логіна/пароля», що дозволяє вносити зміни до запису логіна та пароля у веб-додатку.

Алгоритм редагування логіну/пароля проходить наступним чином.

Ви натискаєте кнопку «EDIT» для вибраного запису. Відкривається сторінка редагування «EDIT» (рис. 3.7), де можна змінювати поля, пов'язані з ім'ям, логіном та паролем та вносити необхідні зміни до полів.



EDIT PASSWORD

Title
twitter

Title account
Account login (optional)

Password
12345678

CANCEL **SAVE**

Рисунок 3.7 – Інтерфейс сторінки «EDIT»

Після завершення редагування Ви зберігаєте внесені зміни, натискаючи кнопку «SAVE». Зміни вступають в дію, і дані про ім'я, логін та пароль оновлюються згідно з внесеними змінами.

Якщо Ви передумали редагувати логін та пароль, то Ви натискаєте «CANCEL» та повертається назад до сторінки «DASHBOARD».

3.4 Робота з відображенням пароля

Після успішної реєстрації та входу до свого облікового запису, Ви потрапляєте до головної сторінки «DASHBOARD» (рис. 3.5), де знаходиться вже створений Вами список логінів та паролів.

Ви отримуєте доступ до функції «Показ пароля», що дозволяє переглядати схований пароль, пов'язаний з конкретним записом логіна.

Алгоритм перегляду схованого пароля проходить наступним чином.

Ви натискаєте кнопку «REVEAL» для вибраного запису. Пароль, пов'язаний з обраним записом логіна, починає Вам відображатися (рис. 3.8).

Після закінчення перегляду паролю Ви можете закрити відображення паролю, натиснувши кнопку «HIDE».

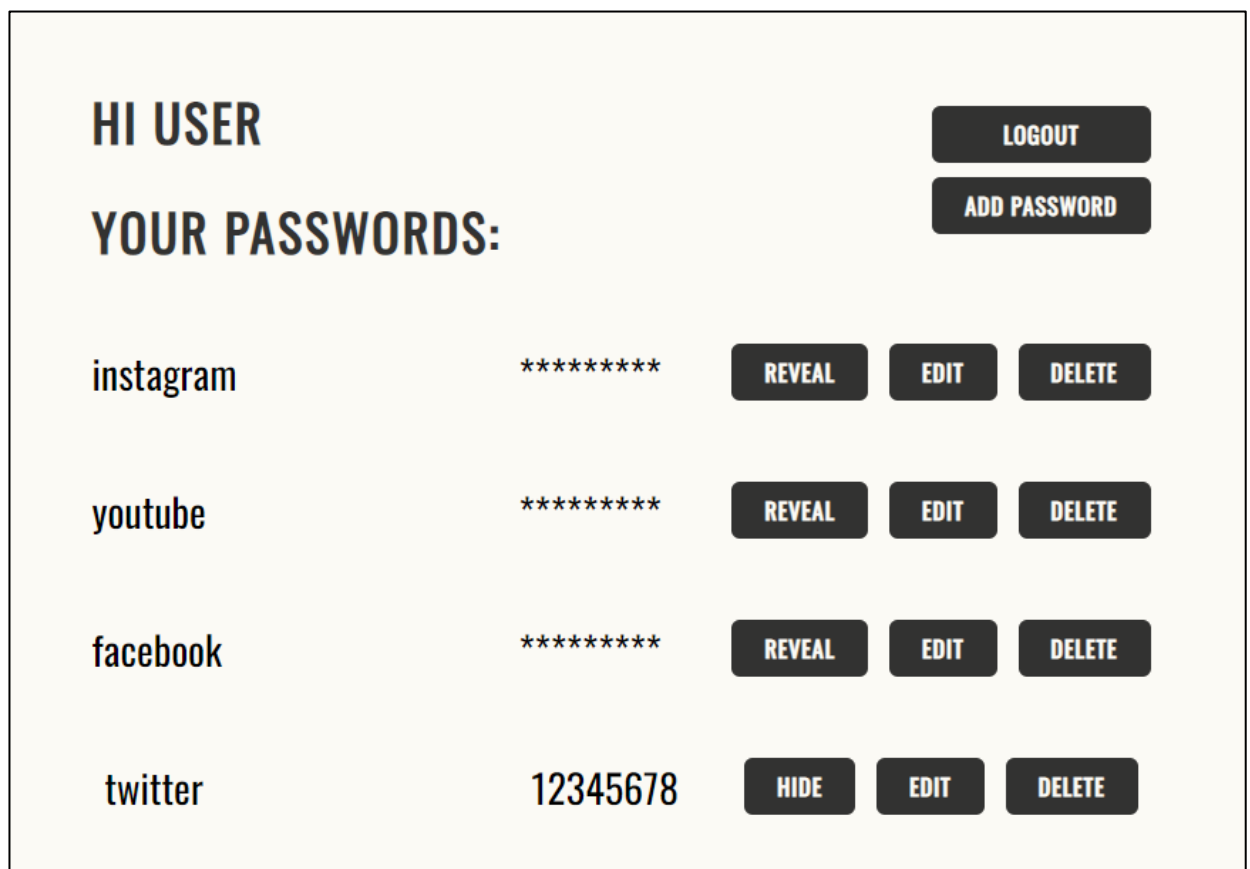


Рисунок 3.8 – Інтерфейс сторінки «DASHBOARD» при перегляді паролю

3.5 Робота з видаленням логіна/пароля

Після успішної реєстрації та входу в свій обліковий запис, Ви побачите головну сторінку «DASHBOARD» (рис. 3.5), де знаходяться вже створені Вами логіни та паролі.

Ви отримуєте доступ до функції «Видалення логіна/пароля», що дозволяє видаляти непотрібні записи логінів та паролів з веб-додатку.

Алгоритм перегляду схованого пароля проходить наступним чином.

Ви натискаєте кнопку «DELETE» для вибраного запису. Відкривається сторінка для видалення паролів «DELETE» (рис. 3.9), де Ви підтверджує видалення натиснувши кнопку «DELETE» Далі відбувається фактичне видалення запису логіна/пароля з бази даних.

Якщо Ви передумали видаляти запис, то Ви натискаєте «CANCEL» та повертаєтесь назад до сторінки «DASHBOARD».

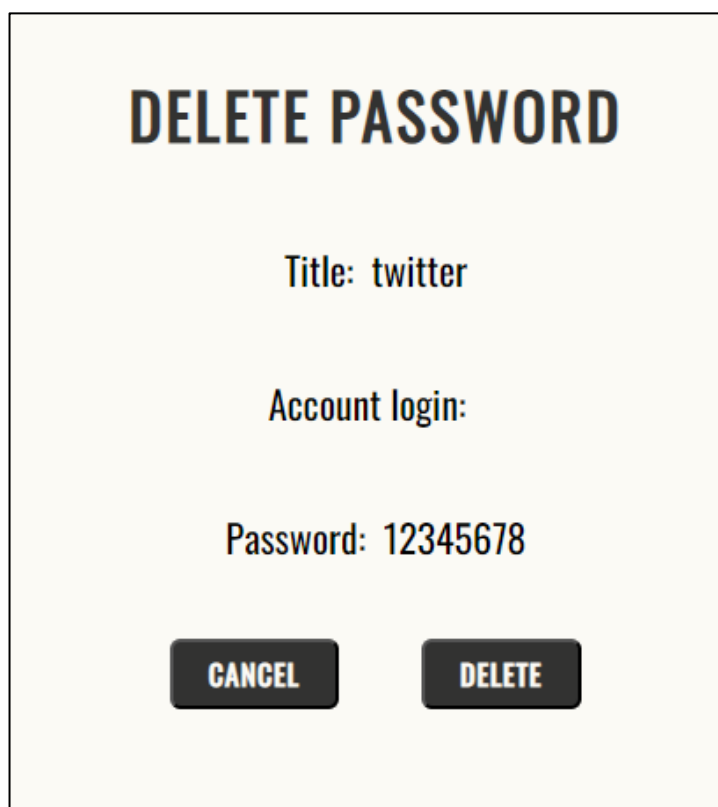


Рисунок 3.9 – Інтерфейс сторінки «DELETE»

3.6 Вихід зі свого облікового запису

Після успішної реєстрації та входу до свого облікового запису, Ви потрапляєте до головної сторінки «DASHBOARD» (рис. 3.5), де знаходиться вже створений Вами список логінів та паролів.

Ви отримуєте доступ до функції «Вийти з облікового запису», що дозволяє Вам вийти зі свого облікового запису. Після виходу Ви втрачаєте доступ до функціоналу програми, який був доступний в обліковому записі.

Алгоритм виходу з облікового запису проходить наступним чином.

Ви натискаєте кнопку «LOGOUT». Після система завершує сеанс облікового запису, видаляє всі дані про користувача з поточної сесії і переключає користувача до сторінки «SIGN UP» (рис. 3.1). У разі потреби Ви можете знову увійти в свій обліковий запис.

ВИСНОВКИ

Під час розробки курсового проєкту мною були пройдені всі шляхи розробки представленого веб-додатку, починаючи з постановки задачі, закінчуючи тестуванням та налагодженням програми.

У результаті розробки було створено веб-додаток «Менеджер паролів», який дозволяє користувачам зручно та безпечно зберігати свої логіни та паролі від різних соціальних мереж та облікових записів. Використовуючи програму, користувачі можуть зручно і безпечно додавати, редагувати та видаляти паролі, а також створювати облікові записи, щоб мати зручний доступ до паролів з будь-якого місця за умови наявності Інтернету.

SPA підхід дозволяє забезпечити більш плавну та швидку роботу програми, оскільки замість переходу зі сторінки на сторінку кожного разу, ми застосовуємо динамічне оновлення та підвантаження контенту на одній сторінці. Враховуючи ці функціональності, наш веб-додаток надає інтуїтивно зрозумілу, зручну та безпечну роботу з управлінням паролями.

Мною було докладно розглянуті такі аспекти програмування: основи об'єктно-орієнтованого програмування, робота з бекенд частиною за допомогою Node.js, базові принципи програмування на Angular, бази даних MongoDB, середовище тестування для API Postman, і бібліотека для симетричного шифрування рядків CryptoJS.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. CryptoJS – CryptoJS. *CryptoJS – CryptoJS*. URL: <https://cryptojs.gitbook.io/docs/> (дата звернення: 11.05.2023).
2. Wigmore I. What is single-page application (SPA)? | Definition from TechTarget. *WhatIs.com*. URL: <https://www.techtarget.com/whatis/definition/single-page-application-SPA> (дата звернення: 08.05.2023).
3. JSON Web Tokens. *Auth0 Docs*. URL: <https://auth0.com/docs/secure/tokens/json-web-tokens> (дата звернення: 12.05.2023).
4. Induction | USIGN INterceptors Tutorial. *Induction / high performance, open source, Java MVC web application framework*. URL: <http://inductionframework.org/using-interceptors-tutorial.html> (дата звернення: 18.05.2023).
5. Що таке фреймворк (framework)? Пояснюємо простими словами. *Highload.today – медіа для розробників*. URL: <https://highload.today/uk/frejmvorki-u-veb-rozrobtsi-shho-tse-yaki-isnuyut-i-dlya-chogo-potribni/> (дата звернення: 18.05.2023).

ДОДАТОК А

Код програми

Бази даних

Файл mongoose.js

```
const mongoose = require('mongoose')

mongoose.Promise = global.Promise;
mongoose.connect('mongodb://localhost:27017/PasswordManager',
  {useNewUrlParser: true}).then(() => {
    console.log("Connected to MongoDB successfully")
  }).catch((e) => {
    console.log("Error while attempting to connect MongoDB");
    console.log(e)
  });

module.exports = {
  mongoose
}
```

Файл index.js

```
const { List } = require('./List.model');
const { Password } = require('./password.model');
const { User } = require('./user.model')

module.exports = {
  List,
  Password,
  User
}
```

Файл List.model.js

```
const mongoose = require('mongoose')

const ListSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
    minLength: 1,
    trim: true
  },
  titleAccount: {
    type: String,
    required: false,
    minLength: 0,
    trim: true
  },
  password: {
    type: String,
    required: true,
    minLength: 1,
    trim: true
  },
},
```

```

    _userId: {
      type: mongoose.Types.ObjectId,
      required: true,
    },
    togglePassword: {
      type: Boolean,
      required: true,
      minLength: 1,
      trim: true
    },
  },
}))

const List = mongoose.model("List", ListSchema);

module.exports = { List }

```

Файл password.model.js

```

const mongoose = require('mongoose')

const PasswordSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
    minLength: 1,
    trim: true
  },
  _listId: {
    type: mongoose.Types.ObjectId,
    required: true
  }
})

const Password = mongoose.model('Password', PasswordSchema);

module.exports = { Password }

```

Файл user.model.js

```

const mongoose = require('mongoose');
const _ = require('lodash')
const jwt = require('jsonwebtoken');
const crypto = require('crypto')
const bcrypt = require('bcryptjs');

const jwtSecret = "vqm3034857nv3047cq5u75834tv38923viy7qvdf66gdghh634m9845";

const UserSchema = new mongoose.Schema({
  email: {
    type: String,
    required: true,
    minLength: 1,
    trim: true,
    unique: true
  },
  userName: {
    type: String,
    required: true,
    minLength: 1,
    trim: true
  },

```

```

    password: {
      type: String,
      required: true,
      minLength: 8,
    },
    sessions: [{
      token: {
        type: String,
        required: true
      },
      expiresAt: {
        type: Number,
        required: true
      }
    }]
  })
})

UserSchema.methods.toJSON = function () {
  const user = this;
  const userObject = user.toObject();

  return _.omit(userObject, ['password', 'sessions']);
}

UserSchema.methods.generateAccessToken = function () {
  const user = this;
  return new Promise((resolve, reject) => {
    jwt.sign({_id: user._id.toHexString()}, jwtSecret, {expiresIn:
"10000000000m"}, (err, token) => {
      if (!err) {
        resolve(token);
      } else {
        reject();
      }
    })
  })
}

UserSchema.methods.generateRefreshAuthToken = function () {
  return new Promise((resolve, reject) => {
    crypto.randomBytes(64, (err, buf) => {
      if (!err) {
        let token = buf.toString('hex');

        return resolve(token);
      }
    })
  })
}

UserSchema.methods.createSession = function () {
  const user = this;

  return user.generateRefreshAuthToken().then((refreshToken) => {
    return saveSessionToDatabase(user, refreshToken);
  }).then((refreshToken) => {
    return refreshToken;
  }).catch((e) => {
    return Promise.reject('Failed to save session to database.' + e);
  })
}

UserSchema.statics.getJWTSecret = function () {

```

```

    return jwtSecret;
  }

  UserSchema.statics.findByIdAndToken = function (_id, token) {
    const User = this;

    return User.findOne({
      _id,
      'sessions.token': token
    });
  }

  UserSchema.statics.findByCredentials = function (email, password) {
    let User = this;
    return User.findOne({email}).then((user) => {
      if (!user) return Promise.reject();

      return new Promise((resolve, reject) => {
        bcrypt.compare(password, user.password, (err, res) => {
          if (res) resolve(user);
          else {
            reject();
          }
        })
      })
    })
  }

  UserSchema.statics.hasRefreshTokenExpired = (expiresAt) => {
    let secondsSinceEpoch = Date.now() / 1000;
    if (expiresAt > secondsSinceEpoch) {
      return false;
    } else {
      return true;
    }
  }

  UserSchema.pre('save', function (next) {
    let user = this;
    let costFactor = 10;

    if (user.isModified('password')) {
      bcrypt.genSalt(costFactor, (err, salt) => {
        bcrypt.hash(user.password, salt, (err, hash) => {
          user.password = hash;
          next();
        })
      })
    } else {
      next();
    }
  })

  let saveSessionToDatabase = (user, refreshToken) => {
    return new Promise((resolve, reject) => {
      let expiresAt = generateRefreshTokenExpiryTime();

      user.sessions.push({'token': refreshToken, expiresAt});

      user.save().then(() => {
        return resolve(refreshToken);
      }).catch((e) => {
        reject(e);
      })
    })
  }

```

```

    })
  })
}

let generateRefreshTokenExpiryTime = () => {
  let daysUntilExpire = "10";
  let secondsUntilExpire = ((daysUntilExpire * 24) * 60) * 60;
  return ((Date.now() / 1000) + secondsUntilExpire)
}

const User = mongoose.model('User', UserSchema);

module.exports = {User}

```

Бекенд веб-додатку

Файл app.js

```

const express = require('express');
const app = express();

const {mongoose} = require('./db/mongoose')

const bodyParser = require('body-parser')
const {List, Password, User} = require('./db/models/index');
const jwt = require('jsonwebtoken');
const CryptoJS = require('crypto-js');
const key = 'your-secret-key';

app.use(bodyParser.json());

app.use(function (req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept, x-access-token, x-refresh-token");
  res.header("Access-Control-Allow-Methods", "GET, POST, OPTIONS, PUT, DELETE, PATCH");

  res.header(
    'Access-Control-Expose-Headers',
    'x-access-token, x-refresh-token'
  );

  next();
});

let authenticate = (req, res, next) => {
  let token = req.header('x-access-token');

  jwt.verify(token, User.getJWTSecret(), (err, decoded) => {
    if (err) {
      res.status(401).send(err)
    } else {
      req.user_id = decoded._id;
      next();
    }
  })
}

let verifySession = (req, res, next) => {

```

```

let refreshToken = req.header('x-refresh-token');

let _id = req.header('_id');

User.findByIdAndToken(_id, refreshToken).then((user) => {
  if (!user) {
    return Promise.reject({
      'error': 'User not found. Make sure that the refresh token
and user id are correct'
    });
  }
  req.user_id = user._id;
  req.userObject = user;
  req.refreshToken = refreshToken;

  let isSessionValid = false;

  user.sessions.forEach((session) => {
    if (session.token === refreshToken) {
      if (User.hasRefreshTokenExpired(session.expiresAt) === false)
{
        isSessionValid = true;
      }
    }
  });

  if (isSessionValid) {
    next();
  } else {
    return Promise.reject({
      'error': 'Refresh token has expired or the session is
invalid'
    });
  }
}).catch((e) => {
  res.status(401).send(e);
});
}

app.get('/all-passwords', authenticate, (req, res) => {
  List.find({
    _userId: req.user_id
  }).then((lists) => {
    const key = 'your-secret-key'; // replace with your own secret key

    // Decrypt password for each list item using AES algorithm
    const decryptedLists = lists.map(list => {
      const ciphertextPassword = list.password;
      const bytes = CryptoJS.AES.decrypt(ciphertextPassword, key);
      list.password = bytes.toString(CryptoJS.enc.Utf8);
      return list;
    });

    res.send(decryptedLists);
  }).catch((error) => {
    console.error(error);
    res.status(500).send('An error occurred while retrieving the
passwords.');
```

```

  });
});

app.post('/create-password', authenticate, (req, res) => {
  let title = req.body.title;

```



```

    let titleAccount = req.body.titleAccount;
    let plaintextPassword = req.body.password;
    let togglePassword = req.body.togglePassword;

    // Encrypt password using AES algorithm
    const key = 'your-secret-key'; // replace with your own secret key
    const ciphertextPassword = CryptoJS.AES.encrypt(plaintextPassword,
key).toString();

    let newList = new List({
        title,
        titleAccount,
        password: ciphertextPassword, // save encrypted password to database
        _userId: req.user_id,
        togglePassword
    });

    newList.save().then((listDoc) => {
        res.send(listDoc);
    }).catch((error) => {
        console.error(error);
        res.status(500).send('An error occurred while saving the password.');
```

});

```

app.patch('/edit-password/:id', authenticate, (req, res) => {
    const key = 'your-secret-key'; // replace with your own secret key

    // Encrypt password using AES algorithm
    const plaintextPassword = req.body.password;
    const ciphertextPassword = CryptoJS.AES.encrypt(plaintextPassword,
key).toString();

    List.findOneAndUpdate({_id: req.params.id, _userId: req.user_id}, {
        $set: {
            title: req.body.title,
            titleAccount: req.body.titleAccount,
            password: ciphertextPassword, // save encrypted password
            togglePassword: req.body.togglePassword
        }
    }).then(() => {
        res.sendStatus(200);
    }).catch((error) => {
        console.error(error);
        res.status(500).send('An error occurred while updating the
password.');
```

});

```

    });

app.delete('/delete-password/:id', authenticate, (req, res) => {
    const key = 'your-secret-key'; // replace with your own secret key

    List.findOneAndDelete({_id: req.params.id, _userId:
req.user_id}).then((removedListDoc) => {
        // Decrypt password using AES algorithm
        const ciphertextPassword = removedListDoc.password;
        const bytes = CryptoJS.AES.decrypt(ciphertextPassword, key);
        const plaintextPassword = bytes.toString(CryptoJS.enc.Utf8);

        // Update removedListDoc object with decrypted password
        const removedListDocWithDecryptedPassword = Object.assign({},
removedListDoc.toObject(), {password: plaintextPassword});
```

```

        res.send(removedListDocWithDecryptedPassword);
    }).catch((error) => {
        console.error(error);
        res.status(500).send('An error occurred while deleting the
password.');
```

});

```

app.get('/find-password/:id', (req, res) => {
    const key = 'your-secret-key'; // replace with your own secret key

    List.findOne({_id: req.params.id}).then((task) => {
        // Decrypt password using AES algorithm
        const ciphertextPassword = task.password;
        const bytes = CryptoJS.AES.decrypt(ciphertextPassword, key);
        const plaintextPassword = bytes.toString(CryptoJS.enc.Utf8);

        // Update task object with decrypted password
        const taskWithDecryptedPassword = Object.assign({}, task.toObject(),
{password: plaintextPassword});

        res.send(taskWithDecryptedPassword);
    }).catch((error) => {
        console.error(error);
        res.status(500).send('An error occurred while finding the
password.');
```

});

```

app.post('/user/create', (req, res) => {
    let body = req.body;
    let newUser = new User(body);

    newUser.save().then(() => {
        return newUser.createSession();
    }).then((refreshToken) => {
        return newUser.generateAccessAuthToken().then((accessToken) => {
            return {accessToken, refreshToken}
        })
    }).then((authToken) => {
        res
            .header('x-refresh-token', authToken.refreshToken)
            .header('x-access-token', authToken.accessToken)
            .send(newUser)
    }).catch((e) => {
        res.status(400).send(e);
    })
})

app.post('/user/login', (req, res) => {
    let email = req.body.email;
    let password = req.body.password;
    let userName = req.body.userName;

    User.findByCredentials(email, password).then((user) => {
        return user.createSession().then((refreshToken) => {
            return user.generateAccessAuthToken().then((accessToken) => {
                return {accessToken, refreshToken}
            })
        });
    }).then((authTokens) => {
        res
            .header('x-refresh-token', authTokens.refreshToken)
            .header('x-access-token', authTokens.accessToken)

```

```

        .send(user)
    })
  }).catch((e) => {
    res.status(400).send(e);
  })
})

app.get('/users/me/access-token', verifySession, (req, res) => {
  req.userObject.generateAccessAuthToken().then((accessToken) => {
    res.header('x-access-token', accessToken).send({accessToken});
  }).catch((e) => {
    res.status(400).send(e);
  })
})

app.listen(3000, () => {
  console.log("Server is listening on port 3000")
})

```

Фронтенд веб-додатку

Файл app.module.ts

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { DashboardComponent } from './pages/dashboard/dashboard.component';
import { HTTP_INTERCEPTORS, HttpClientModule } from "@angular/common/http";
import { NewListComponent } from './pages/new-list/new-list.component';
import { DeleteListComponent } from './pages/delete-list/delete-list.component';
import { EditListComponent } from './pages/edit-list/edit-list.component';
import { FormsModule, ReactiveFormsModule } from "@angular/forms";
import { LoginPageComponent } from './pages/login-page/login-page.component';
import { WebReqInterceptor } from './web-req.interceptor';
import { RegistrationPageComponent } from './pages/registration-page/registration-page.component';

@NgModule({
  declarations: [
    AppComponent,
    DashboardComponent,
    NewListComponent,
    DeleteListComponent,
    EditListComponent,
    LoginPageComponent,
    RegistrationPageComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    FormsModule,
    ReactiveFormsModule
  ],
  providers: [
    {provide: HTTP_INTERCEPTORS, useClass: WebReqInterceptor, multi: true}
  ]
})

```

```

    ],
    bootstrap: [AppComponent]
  })
  export class AppModule { }

```

Файл app-routing.module

```

import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { DashboardComponent } from '../pages/dashboard/dashboard.component';
import { NewListComponent } from '../pages/new-list/new-list.component';
import { DeleteListComponent } from '../pages/delete-list/delete-list.component';
import { EditListComponent } from '../pages/edit-list/edit-list.component';
import { LoginPageComponent } from '../pages/login-page/login-page.component';
import { RegistrationPageComponent } from '../pages/registration-page/registration-page.component';

const routes: Routes = [
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'new-list', component: NewListComponent },
  { path: 'delete-list/:id', component: DeleteListComponent },
  { path: 'edit-list/:id', component: EditListComponent },
  { path: 'login', component: LoginPageComponent },
  { path: 'registration', component: RegistrationPageComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

Файл auth.service.ts

```

import { Injectable } from '@angular/core';
import { HttpClient, HttpResponse } from '@angular/common/http';
import { WebRequestService } from '../web-request.service';
import { Router } from '@angular/router';
import { shareReplay, tap } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class AuthService {

  constructor(private http: HttpClient, private webService:
WebRequestService, private router: Router) {
  }

  login(email: string, password: string) {
    return this.webService.login(email, password).pipe(
      shareReplay(),
      tap((res: HttpResponse<any>) => {
        this.setSession(res.body._id, res.headers.get('x-access-token'),
res.headers.get('x-refresh-token'))
        console.log("logged in")
      })
    )
  }
}

```

```

    )
  }

  registration(email: string, password: string, userName: string) {
    return this.webService.register(email, password, userName).pipe(
      shareReplay(),
      tap((res: HttpResponse<any>) => {
        this.setSession(res.body._id, res.headers.get('x-access-token'),
res.headers.get('x-refresh-token'))
        console.log("register in")
      })
    )
  }

  logout() {
    this.removeSession()
    this.router.navigate(['/login']);
  }

  getAccessToken() {
    return localStorage.getItem('x-access-token')
  }

  getRefreshToken() {
    return localStorage.getItem('x-refresh-token')
  }

  getUserId() {
    return localStorage.getItem('x-refresh-token')
  }

  setAccessToken(accessToken: string) {
    return localStorage.setItem('x-access-token', accessToken)
  }

  private setSession(userId: string, accessToken: string | null,
refreshToken: string | null) {
    localStorage.setItem('user-id', userId);
    if (typeof accessToken === "string") {
      localStorage.setItem('x-access-token', accessToken);
    }
    if (typeof refreshToken === "string") {
      localStorage.setItem('x-refresh-token', refreshToken);
    }
  }

  private removeSession() {
    localStorage.removeItem('user-id');
    localStorage.removeItem('x-access-token');
    localStorage.removeItem('x-refresh-token');
  }

```

Файл password.service.ts

```

import {Injectable} from '@angular/core';
import {WebRequestService} from "../web-request.service";
import {BehaviorSubject, Observable} from "rxjs";

@Injectable({
  providedIn: 'root'
})

```

```

export class PasswordService {

  private listsSubject$: BehaviorSubject<any>
  public list$: Observable<any>

  constructor(private webReqService: WebRequestService) {
    this.listsSubject$ = new BehaviorSubject<any>(
      this.webReqService.get('all-passwords')
    )
    this.list$ = this.listsSubject$.asObservable()
  }

  createList(title: string, password: string, titleAccount: string,
togglePassword: boolean) {
    return this.webReqService.post('create-password', {title, password,
titleAccount, togglePassword});
  }

  editList(listId: string, title: string, password: string, titleAccount:
string) {
    return this.webReqService.patch(`edit-password/${listId}`, {title,
password, titleAccount});
  }

  getLists() {
    return this.listsSubject$.value
  }

  deleteList(listId: string) {
    return this.webReqService.delete(`delete-password/${listId}`);
  }

  getListById (listId: string) {
    return this.webReqService.get(`find-password/${listId}`);
  }
}

```

Файл web-req.interceptors.ts

```

import { Injectable } from '@angular/core';
import { HttpResponseResponse, HttpHandler, HttpInterceptor, HttpRequest } from
"@angular/common/http";
import { catchError, Observable, switchMap, tap, throwError } from "rxjs";
import { AuthService } from "../auth.service";
import { error } from "@angular/compiler-cli/src/transformers/util";

@Injectable({
  providedIn: 'root'
})
export class WebReqInterceptor implements HttpInterceptor{

  constructor(private authService: AuthService) { }

  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<any> {
    request = this.addAuthHeader(request);
    return next.handle(request).pipe(
      catchError((error: HttpResponseResponse) => {
        console.log(error);

        if(error.status === 401) {
          this.authService.logout();

```

```

        }
        return throwError(error)
    })
  )
}

addAuthHeader(request: HttpRequest<any>) {
const token = this.authService.getAccessToken()

    if (token) {
        return request.clone({
            setHeaders: {
                'x-access-token': token
            }
        })
    }
    return request;
}
}

```

Файл web-request.service.ts

```

import { Injectable } from '@angular/core';
import {HttpClient} from "@angular/common/http";

@Injectable({
  providedIn: 'root'
})
export class WebRequestService {

  readonly ROOT_URL: string;

  constructor(private http: HttpClient) {
    this.ROOT_URL = 'http://localhost:3000'
  }

  get (uri: string) {
    return this.http.get(`${this.ROOT_URL}/${uri}`)
  }

  post (uri: string, payload: object) {
    return this.http.post(`${this.ROOT_URL}/${uri}`, payload)
  }

  patch (uri: string, payload: object) {
    return this.http.patch(`${this.ROOT_URL}/${uri}`, payload)
  }

  delete (uri: string) {
    return this.http.delete(`${this.ROOT_URL}/${uri}`)
  }

  login(email: string, password: string) {
    return this.http.post(`${this.ROOT_URL}/user/login`, {
      email,
      password
    }, {
      observe: 'response'
    });
  }

  register(email: string, password: string, userName: string) {

```

```

    return this.http.post(`${this.ROOT_URL}/user/create`, {
      email,
      password,
      userName
    }, {
      observe: 'response'
    });
  }
}

```

Файл app.component.html

```
<router-outlet></router-outlet>
```

Файл app.component.ts

```

import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'frontend';
}

```

Файл dashboard.component.ts

```

import {Component, OnInit} from '@angular/core';
import {PasswordService} from "../../password.service";
import {ActivatedRoute, Router} from "@angular/router";
import {AuthService} from "../../auth.service";

@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css']
})
export class DashboardComponent implements OnInit {
  userName: string | null;
  lists: any;

  revealList: any

  constructor(private passwordService: PasswordService, private route:
ActivatedRoute,
               private router: Router, private authService: AuthService) {
    this.userName = localStorage.getItem('user-name')
  }

  ngOnInit(): void {

    this.passwordService.getLists().subscribe((lists: any) => {
      this.lists = lists
    })
  }

  deleteList(id: string) {
    this.router.navigate(['/delete-list', id])
  }
}

```



```

    }

    editList(id: string) {
      this.router.navigate(['/edit-list', id])
    }

    togglePassword(id: string) {
      this.passwordService.getListById(id).subscribe(val =>
        this.revealList = val)
      this.revealList.togglePassword = !this.revealList.togglePassword
    }

    logout() {
      this.authService.logout()
    }
  }
}

```

Файл dashboard.component.html

```

<div class="centered-content">
  <div class="passwords-list-container ">
    <div class="title-board">
      <div>
        <h1 class="title ">
          Hi {{userName}} </h1>
        <h1 class="title" *ngIf="lists.length > 0"> your Passwords: </h1>
        <h1 class="title" *ngIf="lists.length == 0"> Add your first Password:
      </h1>
    </div>
    <div class="buttons-block">
      <button class="button-header" (click)="logout()">LOGOUT</button>
      <button routerLink="/new-list">ADD PASSWORD</button>
    </div>
    </div>
    <div class="password-list-element" *ngFor="let list of lists">
      <div class="password-list-element-first-column">
        <p class="password-title">{{list.title}}</p>
        <p class="password-title">{{list.titleAccount}}</p>
        <p *ngIf="!list.togglePassword" class="password-title">*****</p>
        <p *ngIf="list.togglePassword" class="password-title">{{
this.list.password}}</p>
        <!--      <input type="text" *ngIf="list.togglePassword" value="
{{ this.list.password}}">-->
      </div>
      <div class="password-list-element-second-column">
        <button *ngIf="!list.togglePassword" class="reveal-button"
(click)="list.togglePassword=!list.togglePassword">
          REVEAL
        </button>
        <button *ngIf="list.togglePassword" class="reveal-button"
(click)="list.togglePassword=!list.togglePassword">
          HIDE
        </button>
        <button class="edit-button"
(click)="editList(list._id)">EDIT</button>
        <button class="delete-button"
(click)="deleteList(list._id)">DELETE</button>
      </div>
    </div>
  </div>
</div>

```

Файл dashboard.component.css

```
@import url('https://fonts.googleapis.com/css2?family=Oswald&display=swap');

.centered-content {
  display: flex;
  justify-content: center;
  align-items: center;
  margin-top: 5%;
}

.passwords-list-container {
  display: flex;
  flex-direction: column;
}

.title {
  font-family: 'Oswald', sans-serif;
  font-size: 36px;
  text-transform: uppercase;
  letter-spacing: 2px;
  color: #323231;
}

.title-board {
  min-width: 700px;
  display: flex;
  justify-content: space-between;
  align-items: baseline;
}

.button-header {
  margin-bottom: 10px;
}

.buttons-block {
  display: flex;
  flex-direction: column;
  justify-content: flex-end;
}

.list-menu {
  display: flex;
  flex-direction: column;
  margin-top: 15px;
  flex-grow: 1;
}

.list-menu-item {
  font-family: 'Oswald', sans-serif;
  text-transform: uppercase;
  letter-spacing: 1px;
  display: flex;
  align-items: center;
  /*padding: 10px 15px;*/
  /*border-radius: 5px;*/
  width: 100%;
  color: #1c1b1b;
```

```

    font-size: 24px;
    margin-bottom: 5px;
  }

  .password-list-element {
    display: flex;
    justify-content: center;
    align-items: center;
    width: 100%;
  }

  .password-title {
    font-family: 'Oswald', sans-serif;
    font-size: 28px;
    width: 130px;
    margin-right: 20px;
  }

  button {
    font-family: 'Oswald', sans-serif;
    padding-left: 20px;
    padding-right: 20px;
    font-size: 16px;
    letter-spacing: 1px;
    height: 40px;
    font-weight: bold;
    border: solid 3px transparent;
    border-radius: 6px;
    margin-right: 15px;
    color: #FBFAF5;
    background-color: #323231;
  }

  button:hover {
    border: solid 3px #323231;
    color: #323231;
    background-color: #FBFAF5;
  }

  .password-list-element-first-column {
    display: flex;
    justify-content: flex-start;
  }

  .password-list-element-second-column {
    display: flex;
    justify-content: flex-end;
  }

```

Файл delete-list.component.ts

```

import { Component, OnInit } from '@angular/core';

import { ActivatedRoute, Router } from '@angular/router';
import { PasswordService } from '../password.service';

@Component({
  selector: 'app-delete-list',
  templateUrl: './delete-list.component.html',
  styleUrls: ['./delete-list.component.css']
})
export class DeleteListComponent implements OnInit {

```

```

list: any;
listId: any;

constructor(private activatedRoute: ActivatedRoute, private
passwordService: PasswordService, private router: Router) { }

ngOnInit(): void {

    this.listId = this.activatedRoute.snapshot.paramMap.get('id');
    this.passwordService.getListById(this.listId).subscribe(val =>
        this.list = val)
}

deleteList (id: string) {
    this.passwordService.deleteList(id).subscribe((response:any) => {
        this.router.navigate(['/'])
    })
}
}

```

Файл delete-list.component.html

```

<div class="centered-content">
  <div class="modal-box">
    <h1>Delete password</h1>
    <div class="create-box">
      <p>Title: &nbsp;{{this.list.title}}</p>
      <p>Account login: &nbsp;{{this.list.titleAccount}}</p>
      <p>Password: &nbsp;{{this.list.password}}</p>
      <div class="buttons-block">
        <button routerLink="/">Cancel</button>
        <button (click)="deleteList(this.list._id)" >Delete</button></div>
      </div>
    </div>
  </div>
</div>

```

Файл delete-list.component.css

```

@import url('https://fonts.googleapis.com/css2?family=Oswald&display=swap');

.centered-content {
  font-family: 'Oswald', sans-serif;
  margin: auto;
  display: flex;
  justify-content: center;
  align-items: center;
  background-color: #FBFAF5;
}

h1 {
  text-align: left;
  font-family: 'Oswald', sans-serif;
  font-size: 36px;
  text-transform: uppercase;
  letter-spacing: 2px;
  color: #323231;
}

.create-box {

```

```

        display: flex;
        justify-content: center;
        align-items: center;
        flex-direction: column;
    }

    button {
        font-family: 'Oswald', sans-serif;
        padding-left: 20px;
        padding-right: 20px;
        font-size: 16px;
        letter-spacing: 1px;
        color: #FBFAF5;
        height: 40px;
        font-weight: bold;
        border-radius: 6px;
        margin-top: 20px;
        background-color: #323231;
        text-transform: uppercase;
    }

    p {
        font-size: 22px;
    }

    .buttons-block {
        width: 100%;
        display: flex;
        justify-content: space-around;
    }

    button:hover {
        border: solid 3px #323231;
        color: #323231;
        background-color: #FBFAF5;
    }

```

Файл edit-list.component.ts

```

import { Component, OnInit } from '@angular/core';
import { PasswordService } from "../../password.service";
import { ActivatedRoute, Router } from "@angular/router";
import { FormControl, FormGroup, Validators } from "@angular/forms";

@Component({
    selector: 'app-edit-list',
    templateUrl: './edit-list.component.html',
    styleUrls: ['./edit-list.component.css']
})
export class EditListComponent implements OnInit {
    show: boolean = false;
    list: any;
    listId: any;
    editListForm: FormGroup | any;

    constructor(private activatedRoute: ActivatedRoute, private
passwordService: PasswordService, private router: Router) { }

    ngOnInit(): void {
        this.listId = this.activatedRoute.snapshot.paramMap.get('id');
        this.passwordService.getListById(this.listId).subscribe(val =>

```

```

        this.list = val)

    this.editListForm = new FormGroup({
        title: new FormControl(this.list.title, Validators.required),
        titleAccount: new FormControl(this.list.titleAccount),
        password: new FormControl(this.list.password, Validators.required)
    })
}

password() {
    this.show = !this.show;
}

editList(title: string, password: string, titleAccount: string) {
    this.passwordService.editList(this.listId, title, password,
    titleAccount).subscribe((response:any) => {

    })
}
}

```

Файл edit-list.component.html

```

<div class="centered-content">
    <form class="modal-box">
        <h1>EDIT PASSWORD</h1>
        <div class="create-box">
            <p>
<label>Title</label><br>
                <input #listTitleInput class="input" type="text"
placeholder="Title" value="{{this.list.title}}" required>
            </p>
            <p>
                <label>Title account</label><br>
                <input #listAccountInput class="input" placeholder="Account login
(optional)" type="text" value="{{this.list.titleAccount}}">
            </p>
            <p>
                <label>Password</label><br>
                <input #listPasswordInput class="input" type="text" placeholder="Edit
password" value="{{this.list.password}}" required>
            </p>
            <div class="buttons-block">
                <button class="button" routerLink="/">Cancel</button>
                <button class="button" (click)="editList(listTitleInput.value,
listPasswordInput.value, listAccountInput.value)"
routerLink="/">Save</button>
            </div>
        </div>
    </form>
</div>

```

Файл edit-list.component.css

```

@import url('https://fonts.googleapis.com/css2?family=Oswald&display=swap');

.centered-content {
    font-family: 'Oswald', sans-serif;
    margin: auto;
    display: flex;
    justify-content: center;

```

```

    align-items: center;
  }

  h1 {
    text-align: center;
    font-family: 'Oswald', sans-serif;
    font-size: 36px;
    text-transform: uppercase;
    letter-spacing: 2px;
    color: #323231;
  }

  .create-box {
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
  }

  input {
    font-family: 'Oswald', sans-serif;
    width: 300px;
    height: 40px;
    border: 1px solid #323231;
    border-radius: 6px;
    font-size: 18px;
    padding-left: 10px;
    background-color: #FBFAF5;
  }

  .button {
    font-family: 'Oswald', sans-serif;
    padding-left: 20px;
    padding-right: 20px;
    font-size: 16px;
    letter-spacing: 1px;
    color: #FBFAF5;
    height: 40px;
    font-weight: bold;
    border-radius: 6px;
    margin-top: 20px;
    background-color: #323231;
    text-transform: uppercase;
  }

  .button:hover {
    border: solid 3px #323231;
    color: #323231;
    background-color: #FBFAF5;
  }

  .buttons-block {
    width: 100%;
    display: flex;
    justify-content: space-around;
  }

```

Файл login-page.component.ts

```

import {Component, OnInit} from '@angular/core';
import {AuthService} from "../../auth.service";
import {HttpResponse} from "@angular/common/http";

```

```

import {Router} from "@angular/router";
import {FormControl, FormGroup, Validators} from "@angular/forms";

@Component({
  selector: 'app-login-page',
  templateUrl: './login-page.component.html',
  styleUrls: ['./login-page.component.css']
})
export class LoginPageComponent implements OnInit {
  newLoginForm: FormGroup | any;
  constructor(private authService: AuthService, private router: Router) {
  }

  ngOnInit(): void {
    this.newLoginForm = new FormGroup({
      email: new FormControl(null, Validators.email),
      password: new FormControl(null, Validators.minLength(8))
    })
  }

  onSubmit() {
    this.authService.login(this.newLoginForm.value.email,
    this.newLoginForm.value.password).subscribe((res: HttpResponse<any>) => {
      this.router.navigate(['/dashboard'])
    })
  }

  onLoginButtonClicked(email: string, password: string) {
    this.authService.login(email, password).subscribe((res:
    HttpResponse<any>) => {
      this.router.navigate(['/dashboard'])
    })
  }
}

```

Файл login-page.component.html

```

<div class="centered-content">
  <form class="modal-box" [formGroup]="newLoginForm">
    <h1>Sign in</h1>
    <div class="create-box">
      <p>
        <span style="color: #DF5757"
          *ngIf="!newLoginForm.get('email').valid">*The entered email is
not a valid</span><br>
        <input class="input" type="text" placeholder="Email"
formControlName="email" required>
      </p>
      <p>
        <span style="color: #DF5757"
          *ngIf="!newLoginForm.get('password').valid">*Password must have
at least 8 symbols</span><br>
        <input class="input" type="password" placeholder="type here password"
formControlName="password" required>
      </p>
      <div class="sign-block">
        <button [disabled]="newLoginForm.invalid" (click)="onSubmit()">Sign
in</button>
        <a routerLink="/registration">Create new account</a></div>
    </div>
  </form>

```



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

Файл login-page.component.css

```
@import url('https://fonts.googleapis.com/css2?family=Oswald&display=swap');

.centered-content {
  font-family: 'Oswald', sans-serif;
  margin: auto;
  display: flex;
  justify-content: center;
  align-items: center;
}

h1 {
  text-align: left;
  font-family: 'Oswald', sans-serif;
  font-size: 36px;
  text-transform: uppercase;
  letter-spacing: 2px;
  color: #323231;
}

.create-box {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
}

input {
  font-family: 'Oswald', sans-serif;
  width: 300px;
  height: 40px;
  border: 1px solid #323231;
  border-radius: 6px;
  font-size: 18px;
  padding-left: 10px;
  background-color: #FBFAF5;
}

button {
  font-family: 'Oswald', sans-serif;
  padding-left: 20px;
  padding-right: 20px;
  font-size: 16px;
  letter-spacing: 1px;
  color: #FBFAF5;
  height: 40px;
  font-weight: bold;
  border-radius: 6px;
  margin-top: 20px;
  background-color: #323231;
  text-transform: uppercase;
}

button:hover {
  border: solid 3px #323231;
  color: #323231;
  background-color: #FBFAF5;
}
```

```

}

a {
  color: #323231;
  margin-top: 15px;
}

.sign-block {
  display: flex;
  flex-direction: column;
}

```

Файл new-list.component.ts

```

import {Component, OnInit} from '@angular/core';
import {PasswordService} from "../../password.service";
import {Router} from "@angular/router";
import {FormBuilder, FormControl, Validators} from "@angular/forms";
import {FormGroup} from "@angular/forms";

@Component({
  selector: 'app-new-list',
  templateUrl: './new-list.component.html',
  styleUrls: ['./new-list.component.css']
})
export class NewListComponent implements OnInit {

  newListForm: FormGroup | any;

  constructor(private passwordService: PasswordService, private router:
Router) {
  }

  ngOnInit(): void {
    this.newListForm = new FormGroup({
      title: new FormControl(null, Validators.minLength(1)),
      titleAccount: new FormControl(null),
      password: new FormControl(null, Validators.minLength(1))
    })
  }

  onSubmit() {
    this.passwordService.createList(this.newListForm.value.title,
      this.newListForm.value.password,
      this.newListForm.value.titleAccount, false).subscribe((response: any)
=> {
      this.router.navigate([''])
    })
  }
}

```

Файл new-list.component.html

```

<div class="centered-content">
  <form class="modal-box" [formGroup]="newListForm" (ngSubmit)="onSubmit()">
    <h1>CREATE A NEW PASSWORD</h1>
    <div class="create-box">
      <p>
        <label style="color: #DF5757" *ngIf="!newListForm.get('title').valid"
for="userName">*The title shouldn't be

```

```

        empty</label><br>
        <input id="userName" class="input" type="text" placeholder="Title"
formControlName="title" required >
    </p>
    <p>
        <label style="visibility: hidden" for="userName">*The title shouldn't
be empty</label><br>
        <input class="input" type="text" placeholder="Account login
(optional)" formControlName="titleAccount" >
    </p>
    <p>
        <label style="color: #DF5757"
*ngIf="!newListForm.get('password').valid" for="password">*The password
shouldn't be empty</label><br>
        <input id="password" class="input" type="password"
placeholder="Password" formControlName="password" required >
    </p>
    <div class="sign-block">
        <button routerLink="/" >Cancel</button>
        <input type="submit" [disabled]="newListForm.invalid" class="input-
btn" value="Create">
    </div>
</div>
</form>
</div>

```

Файл new-list.component.css

```

@import url('https://fonts.googleapis.com/css2?family=Oswald&display=swap');

.centered-content {
    font-family: 'Oswald', sans-serif;
    margin: auto;
    display: flex;
    justify-content: center;
    align-items: center;
    background-color: #FBFAF5;
}

h1 {
    text-align: left;
    font-family: 'Oswald', sans-serif;
    font-size: 36px;
    text-transform: uppercase;
    letter-spacing: 2px;
    color: #323231;
}

.create-box {
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
}

.input {
    font-family: 'Oswald', sans-serif;
    width: 300px;
    height: 40px;
    border: 1px solid #323231;
    border-radius: 6px;
    font-size: 18px;
}

```

```

padding-left: 10px;
background-color: #FBFAF5
}

button {
  font-family: 'Oswald', sans-serif;
  padding-left: 20px;
  padding-right: 20px;
  font-size: 16px;
  letter-spacing: 1px;
  color: #FBFAF5;
  height: 40px;
  font-weight: bold;
  border-radius: 6px;
  margin-top: 20px;
  background-color: #323231;
  text-transform: uppercase;
}

.input-btn {
  font-family: 'Oswald', sans-serif;
  padding-left: 20px;
  padding-right: 20px;
  font-size: 16px;
  letter-spacing: 1px;
  color: #FBFAF5;
  height: 40px;
  font-weight: bold;
  border-radius: 6px;
  margin-top: 20px;
  background-color: #323231;
  text-transform: uppercase;
}

.sign-block {
  width: 100%;
  display: flex;
  justify-content: space-around;
}

button:hover {
  border: solid 3px #323231;
  color: #323231;
  background-color: #FBFAF5;
}

.input-btn:hover {
  border: solid 3px #323231;
  color: #323231;
  background-color: #FBFAF5;
}

```

Файл registration-page.component.ts

```

import { Component, OnInit } from '@angular/core';
import { HttpResponse } from '@angular/common/http';
import { AuthService } from '../auth.service';
import { FormControl, FormGroup, Validators } from '@angular/forms';

@Component({
  selector: 'app-registration-page',
  templateUrl: './registration-page.component.html',

```

```

    styleUrls: ['./registration-page.component.css']
  })
  export class RegistrationPageComponent implements OnInit {
    newRegisterForm: FormGroup | any;
    constructor(private authService: AuthService) { }

    ngOnInit(): void {
      this.newRegisterForm = new FormGroup({
        name: new FormControl('', Validators.minLength(2)),
        email: new FormControl('', [Validators.minLength(1),
Validators.email]),
        password: new FormControl('', Validators.minLength(8))
      })
    }

    onSubmit() {
      this.authService.registration(this.newRegisterForm.value.email,
this.newRegisterForm.value.password,
this.newRegisterForm.value.name).subscribe((res: HttpResponse<any>) => {
        localStorage.setItem('user-name', this.newRegisterForm.value.name);

        console.log(res);
      })
    }

    onRegistrationButtonClicked(email: string, password: string, userName:
string) {
      this.authService.registration(email, password, userName).subscribe((res:
HttpResponse<any>) => {
        localStorage.setItem('user-name', userName);
        console.log(res);
      })
    }
  }
}

```

Файл registration-page.component.html

```

<div class="centered-content">
  <form class="modal-box" [formGroup]="newRegisterForm">
    <h1>Sign up</h1>
    <div class="create-box">
      <p>
        <label style="color: #DF5757"
*ngIf="!newRegisterForm.get('name').valid" for="userName">*The name shouldn't
be empty</label><br>
        <input id="userName" class="input" type="text" placeholder="Name"
formControlName="name" required>
      </p>
      <p>
        <label style="color: #DF5757"
*ngIf="!newRegisterForm.get('email').valid" for="email">*The entered email is
not a valid</label><br>
        <input id="email" class="input" type="text" placeholder="Email"
formControlName="email" required>
      </p>
      <p>
        <label style="color: #DF5757"
*ngIf="!newRegisterForm.get('password').valid" for="password">*Password must
have at least 8 symbols</label><br>
        <input id="password" class="input" type="password"
placeholder="Password" formControlName="password" required>
      </p>
    </div>
  </form>

```

```

    <div class="sign-block">
        <button routerLink="/login" [disabled]="newRegisterForm.invalid"
(click)="onSubmit() ">Sign up</button>
        <a routerLink="/login">Do you already have an account?</a></div>
    </div>
</form>
</div>

```

Файл registration-page.component.css

```

@import url('https://fonts.googleapis.com/css2?family=Oswald&display=swap');

.centered-content {
    font-family: 'Oswald', sans-serif;
    margin: auto;
    display: flex;
    justify-content: center;
    align-items: center;
    background-color: #FBFAF5;
}

h1 {
    text-align: left;
    font-family: 'Oswald', sans-serif;
    font-size: 36px;
    text-transform: uppercase;
    letter-spacing: 2px;
    color: #323231;
}

.create-box {
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
}

input {
    font-family: 'Oswald', sans-serif;
    width: 300px;
    height: 40px;
    border: 1px solid #323231;
    border-radius: 6px;
    font-size: 18px;
    padding-left: 10px;
    background-color: #FBFAF5;
}

button {
    font-family: 'Oswald', sans-serif;
    padding-left: 20px;
    padding-right: 20px;
    font-size: 16px;
    letter-spacing: 1px;
    color: #FBFAF5;
    height: 40px;
    font-weight: bold;
    border-radius: 6px;
    margin-top: 20px;
    background-color: #323231;
    text-transform: uppercase;
}

```

```
}  
  
a {  
  color: #323231;  
  margin-top: 15px;  
}  
  
.sign-block {  
  display: flex;  
  flex-direction: column;  
}  
  
button:hover {  
  border: solid 3px #323231;  
  color: #323231;  
  background-color: #FBFAF5;  
}
```