КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

РОЗРОБКА ВЕБ-ПРОЄКТУ ІЗ ЗАСОБАМИ ШИФРУВАННЯ ДАНИХ

Виконав студент 4-го курсу
Шимків Дмитро Іванович

(підпис)

Науковий керівник:
доцент, кандидат фізико-математичних наук
Кузенко Володимир Федорович

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«10» червня 2022 р.,
протокол № 10
Завідувач кафедри
М. С. Нікітченко

(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи 42 сторінки, 16 ілюстрацій, 9 джерел посилання, 4 таблиці

WEB-ПРОЄКТ, КЛІЄНТСЬКА ЧАСТИНА, СЕРВЕРНА ЧАСТИНА, БАЗА ДАНИХ, ШИФРУВАННЯ, РОЗШИРЕННЯ, МЕНЕДЖЕР ПАРОЛІВ, JWT, ДВОФАКТОРНА АВТОРИЗАЦІЯ.

Об’єктом роботи є процес збереження та перегляду паролів, шифрування нотаток та зручний спосіб для використання створених даних. Предметом роботи є вебсайт для простого та ефективного виконання процесу, описаного вище.

Метою роботи є розробка клієнтської та серверної частини, а також розширення для браузера Google Chrome.

Інструменти розробки: мова програмування JavaScript, мова програмування TypeScript, мова розмітки сторінок HTML, бібліотека ReactJs для створення інтерфейсу, бібліотека компонентів MaterialUI, бібліотека для кращої взаємодії із API запитами Redux Toolkit, фреймворк для створення серверної частини ExpressJs, нереляційна база даних MongoDB, редактор коду Visual Studio Code, застосунок для перегляду даних MongoDB Compass, застосунок для тестування REST API Postman.

Результати роботи: розроблено web-проєкт із клієнтською та серверною частиною для зберігання паролів та зашифрованих нотаток. Також було створено розширення для зручнішого використання застосунку. Вебсайт містить двофакторну авторизацію з використанням паролю та шестизначного коду, надісланого в SMS.

Результат роботи можуть використовувати всі, хто має на меті та вважає корисним збереження паролів та нотаток в одному місці з подальших зручним використанням збережених даних.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об'єкта розробки. У сучасному світі нові додатки, які є корисними, з'являються досить часто. У більшості з них задля надання користувачу особистого досвіду використовується авторизація за допомогою паролю. Всі ці паролі необхідно пам'ятати, оскільки використання однакових секретних ключів є досить небезпечним. Звичайно, їх можна відновити, якщо забув. Проте це займе певний час. Деякі застосунки для авторизації використовують не звичайний рядок для паролю, а якийсь текст. Наприклад, гаманці для зберігання криптовалюти використовують текст із 12 слів для відновлення. Враховуючи наведені приклади, постає актуальна проблема збереження секретних даних.

Актуальність роботи та підстави для її виконання. Проаналізувавши вище сказане, можна зробити висновок, що багато користувачів хотіли б зберігати свою секретну інформацію в одному місці. Воно має бути добре захищене та зручне у використанні. Таким місцем може стати web-застосунок, який буде забезпечувати необхідну функціональність.

Мета й завдання роботи. Метою роботи є розробка web-проєкту, який буде складатись із клієнтської та серверної частин, а також буде містити розширення для браузера Google Chrome. Для досягнення цієї мети було поставлено наступні завдання:

- Вибрати технології та інструменти, які будуть використовуватись для розробки
- Розробити систему двофакторної авторизації
- Спроектувати структуру даних, яка буде зберігатись в БД
- Побудувати REST API для обміну даними між клієнтом та сервером
- Розробити користувацький інтерфейс
- Забезпечити секретність необхідних даних

Об'єкт, методи й засоби розроблення. Об'єктом розроблення є процес збереження та переглядання паролів, шифрування нотаток та зручний спосіб для використання створених даних. Предметом роботи є вебсайт для простого та ефективного виконання вищеописаного процесу.

Перед початком виконання роботи було обрано ряд інструментів та технологій, які використовуються для розробки веб-проєктів.

Під час розробки було використано каскадну модель програмування. Або ж по іншому модель waterfall. Ця модель має таку назву, оскільки її діаграма має вигляд схожий на водоспад. Її суть полягає в тому, що кожен наступний етап залежить від попереднього й розпочинається лише тоді, коли його повністю закінчено. Особливість такої моделі програмування полягає в тому, що після завершення етапу, до нього більше не повертаються. Так при розробці було повністю спроектовано та побудовано серверну частину із REST API. Лише після цього було розпочато процес розробки користувацького інтерфейсу. [1]

Для створення застосунку було використано мови програмування TypeScript та JavaScript, фреймворк Express, бібліотеку для побудови інтерфейсів React, мови розмітки та стилів HTML, CSS. Також було застосовано ряд додаткових бібліотек для шифрування даних, кодування та валідації токена, відправлення та перевірки SMS, а також деякі допоміжні для клієнтської частини.

Можливі сфери застосування. Розроблений застосунок може використовуватись усіма, хто бажає тримати свої паролі та секретні нотатки в одному зручному та захищеному місці.

РОЗДІЛ 1 ОГЛЯЛ ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

Перед тим, як розпочинати процес розробки продукту було вибрано набір інструментів та технологій, які використовуються при побудові комплексних веб-проєктів. Для розробки практичної частини роботи було вибрано набір MERN. Це скорочення від перших літер основних інструментів. А саме: MongoDB, Express, React та Node. Також є варіація цього набору із використанням фреймворку Angular для розробки клієнтської частини (MEAN).

1.1 Огляд засобів для розробки серверної частини

Для розробки серверної частини проєкту було використано наступні технології та інструменти:

Мова програмування **JavaScript.** Основним інструментом при побудові продукту є мова програмування. Для розробки серверної частини було використано мову програмування JavaScript. JS – динамічна, об'єктно-орієнтована прототипна мова програмування. Частіше вона використовується для взаємодії з користувачем на клієнтській стороні програми, проте все частіше її починають також використовувати для побудови серверної частини. Перевагами використання цієї мови програмування є її швидкість, простота та популярність.

Фреймворк **Express.** Є вільним, відкритим програмним забезпеченням, яке використовується для побудови серверної частини веб-проєктів. Express є швидким, гнучким та мінімалістичним фреймворком для додатків Node.js [2]. Він надає ряд зручних функцій, які полегшують розробку та відкривають нові можливості. Однією з таких нових можливостей є створення middleware. Це

функція, яка має доступ до об’єктів запиту та відповіді, а також до наступної функції в циклі. Її використання може бути досить корисним.

Нереляційна СКБД **MongoDB.** Це документо-орієнтована система керування базами даних з відкритим вихідним кодом, яка не потребує опису схеми таблиць. Дані в MongoDB зберігаються у форматі JSON. Із можливостей цієї системи можна також виділити гнучку мову формування запитів, динамічні запити, профілювання запитів, швидкі оновлення та ефективне зберігання бінарних даних великих обсягів. [3] Для збереження даних у хмарі використовується сервіс MongoDB Atlas. Він є безкоштовним для особистого використання.

Бібліотека **Mongoose.** Ця бібліотека використовується для створення з’єднання між MongoDB та Express. Mongoose забезпечує просте рішення для моделювання даних на основі схеми. Вона включає в себе вбудоване переведення типів, перевірку, побудову запитів та багато іншого. [4]

Бібліотека **bcrypt**. Ця бібліотека надає криптографічну функцію формування ключа, що використовується для безпечного зберігання паролів. За її допомогою можна зберігати в базі зашифрований ключ та пізніше порівнювати зі справжнім паролем при авторизації.

Бібліотека **crypto-js**. Є досить популярною бібліотекою для шифрування, дешифрування на основі заданого секретного ключа. Містить в собі багато криптографічних алгоритмів.

Бібліотека **jsonwebtoken**. JWT (JSON Web Token) – це стандарт токена доступу на основі JSON. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який надалі використовує їх для підтвердження своєї особи. Бібліотека jsonwebtoken надає всі необхідні для цього функції. [5]

Бібліотека **twilio**. В даній роботі ця бібліотека використовувалась для відправлення та перевірки SMS із кодом для двофакторної авторизації. Проте, вона надає багато іншого функціоналу, який може бути корисним.

MongoDB Compass – графічний клієнт, який використовується для взаємодії з базою даних. У ній також присутній термінал для роботи з даними. З нього можна виконувати всі основні операції, такі як: видалення, створення, оновлення та інші.

Postman – програма, яка використовується для тестування API запитів. Вона дозволяє виконати запити до сервера та переглянути результати їх виконання.

1.2 Огляд засобів для розробки клієнтської частини

Для розробки клієнтської частини проєкту було використано такі технології та інструменти:

Мова програмування **TypeScript**. Ця мова програмування є зворотно сумісною з мовою JavaScript. Тобто кожен код, який написаний на JavaScript валідний у TypeScript, але не навпаки. Ця мова має ряд переваг завдяки яким все частіше їй надають перевагу у виборі інструмента для розробки. Основною перевагою TypeScript(TS) у порівнянні з JavaScript(JS) є статична типізація. Тобто, на відміну від JS, у TS можна явно вказувати типи. Основними типами є: boolean, number, string, symbol, null, undefined, а також можна використовувати any, щоб вказати будь-який тип. Для створення власних типів використовуються такі ключові слова: type, interface, enum. Також однією з переваг цієї мови є підтримка використання повноцінних класів, як в традиційних об’єктно-орієнтовних мовах. Оскільки сучасні браузери поки не розпізнають TypeScript, він повинен бути скомпільований у звичайний JavaScript.

Бібліотека **React.** Це відкрита бібліотека, розроблена Facebook для полегшення створення односторінкових застосунків (Single Page Application). Особливістю таких застосунків є те, що сторінка не перезавантажується, а змінює свій вміст. React дозволяє швидко та зручно створювати великі веб-проєкти. Програми, які побудовані на цій бібліотеці складаються з компонентів, що в свою чергу містять JSX синтаксис. Він дозволяє писати вирази мови HTML прямо у .js файлах. Також однією з ключових особливостей React є підтримка віртуального DOM. Це легка копія реального DOM дерева. Всі зміни стану програми спочатку записуються до віртуального, який потім порівнюються з реальним. Тоді в реальний DOM записуються лише змінені частини. Це значно підвищує швидкість роботи програми. Для відслідковування стану програми існують зручні інструменти. Одним з таких інструментів є React Developer Tools. Це розришення для браузера, яке надає зручний інтерфейс з яким процес розробки стає значно легшим. React є одним із найпопулярніших рішень для створення клієнтської частини програми.

Біблотека **Material UI**. Вона надає готові компоненти, які використовуються для побудови користувацького інтерфейсу. Це значно пришвидшує розробку та надає можливість сконцентруватися в основному на розробці логіки програми. У бібліотеці реалізовано багато основних компонентів, які використовуються при побудові веб-проєктів. Такі як кнопки, поля вводу, випадаючі списки, меню та інші. Компоненти досить легко стилізуються за потреби. Також Material UI надає досить багато іконок, які можна використовувати у своїх проєктах.

Бібіліотека **Redux Toolkit**. Ця бібліотека є надбудовою над звичайним Redux. Вона забезпечує легку взаємодію з даними. Вони зберігаються в одному місці, й можуть бути використанні у будь-якому місці програми. Також однією з можливостей бібліотеки є зручна взаємодія із запитами до сервера. Усі відповіді кешуються. Новий запит відправляється лише тоді, коли змінюються вхідні параметри. В іншому випадку просто повертається попереднє значення. При запиті на оновлення даних або їх створення, можна оновити дані для

закешованого запиту ще до того, як він буде успішно виконаний. Для користувача всі дії виглядають моментальними.

Бібліотека **Draft js.** Це фреймворк для створення текстових редакторів для React. Draft.js дозволяє створювати будь-який тип форматованого введення тексту, незалежно від того, чи потрібно додати лише кілька стилів тексту або ж створити складний текстовий редактор. [6]

Бібліотека **React draft wysiwyg.** Це бібліотека, яка побудована із використанням React та Draft. Та надає зручний компонент текстового редактора, який легко змінюється під власні потреби.

Бібліотека **React toastify.** Дуже проста у використанні та дає змогу організувати роботу сайту в гарному вигляді. Вона використовується для показу спливаючих повідомлень. Бібліотека надає різні типи цих повідомлень, такі як: успіх, попередження, помилка та інші.

РОЗДІЛ 2 СЕРВЕРНА ЧАСТИНА ПРОЄКТУ

Наступним етапом після вибору технологій була розробка серверної частини проєкту. Було спроєктовано схему бази даних та побудовано REST API для обміну даними між клієнтом та сервером.

2.1 Огляд схеми бази даних

У якості бази даних було вибрано нереляційну БД MongoDB. Нереляційні бази даних NoSQL є нетабличними і зберігають дані в інший спосіб, ніж звичайні реляційні таблиці. Вони бувають різних типів. Основними з них є документ, ключ-значення, широкий стовпець і графік. Вони забезпечують гнучкі схеми і легко масштабуються з великими обсягами даних і високими навантаженнями. Бази даних NoSQL були створені у відповідь на обмеження традиційної технології реляційних баз даних. У порівнянні з реляційними, NoSQL часто більш масштабовані та забезпечують високу продуктивність. Крім того, гнучкість і простота використання їх моделей даних можуть прискорити розробку в порівнянні з реляційною моделлю, особливо в середовищі хмарних обчислень. [7] Основні переваги нереляційних БД:

- Можливість обробляти великі обсяги даних на високій швидкості завдяки архітектурі масштабування
- Можливість зберігання неструктурованих, напівструктурованих або структурованих даних
- Просте оновлення схем і полів

Перед розробкою було побудовано схему бази даних. Її діаграму класів показано на рисунку 1.

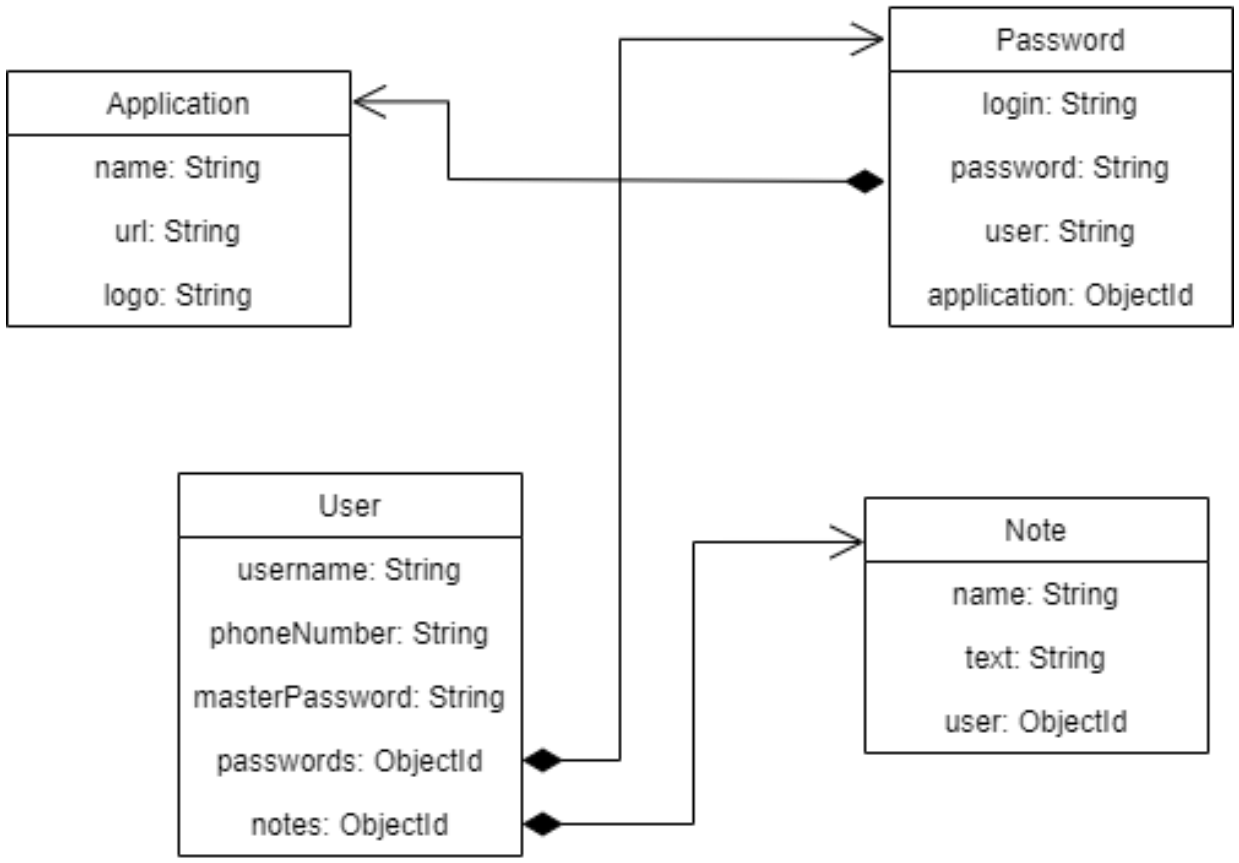

Рисунок 1 - Діаграма класів.

На цій діаграмі ми бачимо наступні сутності:

- User – користувач застосунку;
- Application – програма, яка вказується при створенні паролів.
- Password – паролі користувачів
- Note – зашифровані нотатки

Нижче приведений опис даних у кожній з таблиць.

У таблиці 1 містяться поля для збереження даних про сайти.

Таблиця 1 - Application

Атрибут	**Тип**	**Опис**
name	string	Назва сайту
url	string	Посилання на сайт
logo	string	Логотип сайту

У таблиці 2 показані поля для збереження даних про нотатки користувача.

Таблиця 2 - Note

Атрибут	**Тип**	**Опис**
name	string	Назва нотатки
text	string	Текст нотатки
user	objectId	Ідентифікатор користувача

Таблиця 3 зображує поля для збереження інформації про паролі.

Таблиця 3 - Password

Атрибут	**Тип**	**Опис**
login	string	Логін користувача
password	string	Пароль користувача на сайті
user	objectId	Ідентифікатор користувача
application	objectId	Ідентифікатор сайту

У таблиці 4 містяться поля для збереження інформації про користувача.

Таблиця 4 - User

Атрибут	**Тип**	**Опис**
username	string	Ім’я користувача

phoneNumber	string	Номер телефону
masterPassword	string	Пароль користувача
passwords	objectId	Список ідентифікаторів паролів
notes	objectId	Список ідентифікаторів нотаток

2.2 Огляд доступних запитів

Для обміну даними між клієнтом та сервером було побудовано REST API. Запит із вхідними параметрами відправляється на сервер, там вони обробляються, зберігаються в базу даних, а результат виконання повертається назад до клієнту. Для отримання даних використовується метод GET, щоб додати нові дані потрібно використовувати метод POST, метод PATCH потрібен для оновлення, а для видалення - метод DELETE. Оскільки сервер запускається локально всі запити починаються з “http://localhost:5000/”. Далі розглянемо реалізовані запити.

Всі запити для роботи із сутністю Users починається з ‘/user’. Доступні запити:

- “/auth” – GET запит для отримання інформації про користувача за допомогою токена.
- “/create” – POST запит для створення нового користувача. Вхідні параметри: ім’я, мобільний номер та пароль.
- “/login” – POST – запит для входу користувача. Передається ім’я або номер та пароль.

Верифікація за номером мобільного телефону обробляється за допомогою запитів, які починаються з “/sms”. Доступні запити:

- “/send” – POST запит для відправлення смс. Вхідним параметром є номер телефону, на який воно буде відправлено.

- “/verify” – POST запит для перевірки відправлено коду в смс. Вхідні параметри: номер телефону та 6-ти значний код. Повертається токен, у якому закодовано дані про користувача.

Робота з сутністю Applications виконується за допомогою запитів, що починаються з “/applictions”. Реалізовані наступні запити:

- “/” – GET метод, який повертає всі доданні в базу додатки
- “/create” – POST запит для створення нового додатку. Обов’язковим параметром є назва додатку. Опціональними є логотип та посилання, якщо це вебсайт.

Запити для роботи з паролями починаються з “/password”. До всіх запитів потрібно додати заголовок з токеном авторизації, щоб забезпечити доступ користувача лише до своїх даних. Доступні запити:

- “/” – GET запит для отримання всіх паролів.
- “/find” – POST запит для отримання запитів, відфільтрованих за посиланням на додаток. Вхідними даними є url, за яким відбувається фільтрація.
- “/:id” – GET запит отримання паролю за його id.
- “/create” – POST запит на створення нового паролю. Вхідні параметри: логін, пароль, назва додатку для якого створюється пароль.
- “/delete/:id” – DELETE запит для видалення паролю за id.

Взаємодія із зашифрованими нотатками відбувається за допомогою запитів, які починаються з “/note”. Розглянемо доступні запити:

- “/” – GET запит для отримання всіх нотаток користувача.
- “/create” – POST запит для створення нотатки. Передається назва нотатки, текст та секретний ключ, який використовується для шифрування.
- “/decrypt/:id/:key” – GET запит для отримання розшифрованої нотатки за її id та секретним ключем.
- “/delete/:id” – DELETE запит для видалення нотатки за її id

Після того, як усі запити було розроблено, вони були протестовані в програмі Postman. Вона дозволяє досить легко перевірити коректність виконання запитів. Для цього потрібно задати метод (GET, POST, PATCH, DELETE та інші), url, а також за необхідності тіло запиту.

2.3 Забезпечення надійності програми

Оскільки в базі зберігаються паролі та секретні нотатки, вони мають бути добре захищені. Отже, однією з ключових задач було забезпечення надійного зберігання користувацьких даних та доступу до них. Для її досягнення було поставлено наступні підзадачі:

- Розробити систему двофакторної авторизації
- Забезпечити можливість доступу до даних лише авторизованим користувачам
- Шифрувати паролі та нотатки користувача

Щоб доступ до даних міг отримувати тільки авторизований користувач було розроблено систему, яка відповідала за вхід користувача у програму. Для початку користувач повинен створити аккаунт для подальшого використання. При цьому вказати ім’я, номер мобільного телефону та головний пароль, який буде використовуватися для входу. Пароль зберігається в базі даних у зашифрованому вигляді. Під час входу користувач використовує своє ім’я або номер телефону та пароль. Після чого йому приходить код підтвердження у вигляді смс. І лише ввівши цей код, після його верифікації користувач отримує JWT токен, який генерується на основі секретного ключа. Далі робота із усіма даними здійснюється лише за наявності цього токена. Він передається у заголовку до кожного запиту та валідується на сервері. Якщо ж токен закодований не правильно, повертається помилка. Доступ до читання, запису та видалення має лише авторизований користувач та лише до своїх даних. Всі створенні паролі зберігаються у базу даних у зашифрованому вигляді. Секретним ключем тут виступає головний пароль користувача, який він вказував

при створенні профілю та додатковий ключ, збережений в програмі. Секретні нотатки також шифруються перед записом в базу даних, проте тут секретним ключем є рядок, який користувач вказує при створенні. Цей рядок не записується в базу. Користувач повинен його запам’ятати та використовувати для подальшого читання нотатки. Розглянемо більш детально реалізацію кожної підзадачі.

Двофакторна авторизація. Першим фактором є звичайний пароль, який при створенні перед зберіганням шифрується за допомогою бібліотеки bcrypt. Це досить популярна бібліотека, яка має 900 тисяч скачувань на тиждень. Вона надає зручні функції, які використовуються для перевірки правильності зашифрованого паролю. Спочатку за допомогою функції bcrypt.genSalt генерується “сіль” для подальшого шифрування. Тут вказується числове значення кількості раундів генерації “солі”. Наприклад, 10. Після цього вона використовується у функції bcrypt.hash в яку передається текстовий рядок і повертається хеш запис, який далі зберігається в базу даних. При вході в програму введений користувачем пароль порівнюється із захешованим за допомогою функції bcrypt.compare. Другим фактором є верифікація користувача за кодом надісланим у смс на номер телефону, вказаний при реєстрації. Для цього використовується бібліотека twilio. Вона є дуже потужним та зручним інструментом, який має багато можливостей, але в даній роботі використовувалась лише для верифікації та надсилання коду. Twilio надає користувачам 15 доларів для тестування можливостей. Одна надіслана смс вартує приблизно 30 центів. В обох випадках (при відправці смс та верифікації коду) використовується функція twilio.verify.sercives в яку потрібно передати id створеного сервісу. Для того, щоб його отримати потрібно зайти на їх сайт, та взяти його у розділі сервісів. Або ж створити новий. Далі для відправлення коду використовується функція verifications.create в яку передається номер телефону та спосіб відправлення. В цій роботі використовувався спосіб відправлення “sms”. Після введення користувачем коду, він перевіряється за допомогою функції verificationChecks.create. До неї передається номер телефону та код.

Якщо він введений правильно, користувачу повертається токен, який використовується, далі.

Доступ до даних. Доступ до всіх важливих даних отримують лише авторизовані користувачі. Для цього їм потрібно передавати токен, отриманий після проходження етапу авторизації. Для генерації токену та його валідації використовується бібліотека jsonwebtoken. Щоб закодувати дані у вигляді токена застосовується функція jwt.sign в неї передаються дані, які потрібно зашифрувати, секретний ключ, який зберігається на сервері та час протягом якого токен буде залишатись валідним після його створення. Для верифікації правильності токена використовується функція jwt.verify. У цьому токені зберігається унікальний ідентифікатор користувача, який далі використовується для отримання даних. Щоб забезпечити можливість відправлення запитів лише авторизованим користувачам використовується особливості Express фреймворку, а саме middleware функція. Вона виконується перед виконання будь-яких інших функцій, та має доступ до об’єкта запиту та відповіді. В ній валідується токен і в разі некоректності повертається помилка. Якщо ж токен правильний, то викликається наступна функція у яку додається id користувача взятого із нього.

Шифрування даних. Для реалізації шифрування як і нотаток так і паролів використовується бібліотека crypto-js. Це потужна бібліотека, яка надає досить велику кількість криптографічних алгоритмів. Вона є досить популярною і має 4 мільйони скачувань в тиждень. В даній роботі був використаний алгоритм шифрування AES (Advanced Encryption Standard). Також відомий під назвою Rijndael — це симетричний алгоритм блочного шифрування (розмір блока 128 біт, ключ 128/192/256 біт), він є фіналістом конкурсу і був прийнятий в якості американського стандарту шифрування урядом США. Вибір припав на AES з розрахуванням на широке використання і активний аналіз алгоритму, як це було із його попередником, DES. Державний інститут стандартів і технологій США опублікував попередню специфікацію AES 26 жовтня 2001 року, після

п'ятилітньої підготовки. 26 травня 2002 року його було оголошено стандартом шифрування. [8]

На рисунку 2 показано схему шифрування за допомогою цього алгоритму.

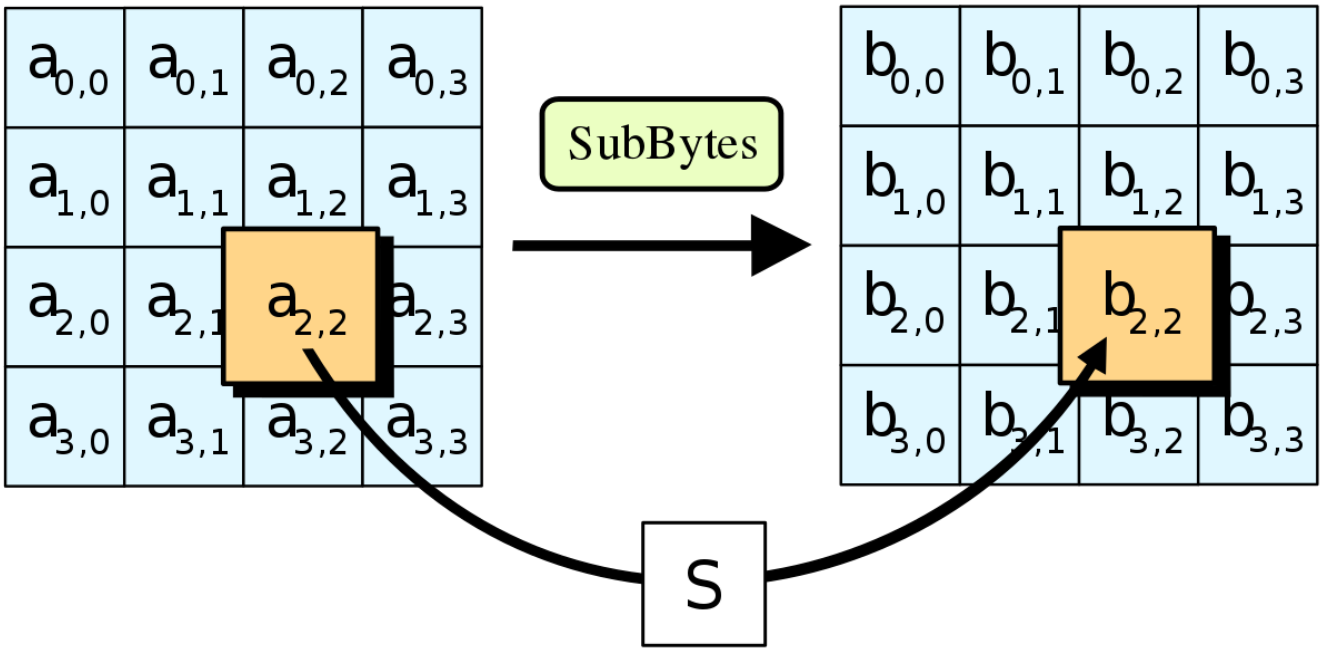

Рисунок 2 - Алгоритм шифрування AES

Для шифрування цим алгоритмом бібліотека crypto-js надає зручні для використання функції. CryptoJS.AES.encrypt використовується для кодування тексту. Сюди передаються дані та секретний ключ. Результат виконання зберігається в базу. Щоб отримати початкове значення використовується обернений алгоритм. Для цього є функція CryptoJS.AES.decrypt.

2.4 Структура проєкту серверної частини

Проєкт складається з наступних основних складових:

- Файл index.js – в ньому відбувається основна ініціалізація, підключення до бази даних та запуск сервера.

- Папка controllers – в ній зберігаються файли, які містять класи для взаємодії з запитами
- Папка middlewares – папка, у якій містяться файли із middleware функціями
- Папка models – класи моделей, які є в базі даних
- Папка routes – функції, які створюють ендпоінти для запитів
- Папка services – в ній містяться класи для взаємодії із БД
- Папка utils – допоміжні функції.

Структура проєкту показана на рисунку 3.

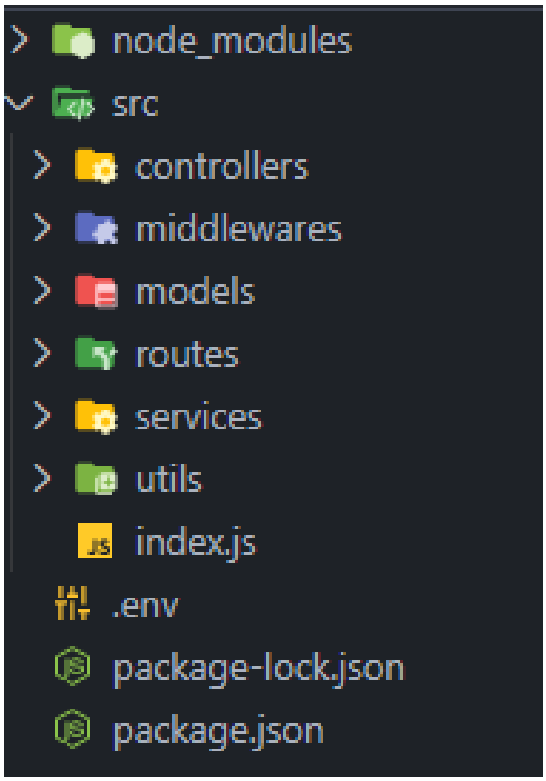

Рисунок 3 - Структура серверної частини

Розглянемо більш детально деякі елементи. Контролери – це класи, які складаються з методів для взаємодії з запитами. Кожен такий метод приймає параметрами два об`єкти: req (запит) та res (відповідь). Вони звертаються до відповідних класів сервісів та у відповідь повертають результат виконання або ж помилку у разі невдачі. Приклад реалізації контролера наведено в додатку А. В

свою чергу класи сервісів містять методи для взаємодії із базою даних. Вони приймають певні параметри, обробляють їх та вносять відповідні зміни в БД. Реалізацію сервісу показано в додатку Б. Завдяки бібліотеці mongoose розробка таких сервісів є досить зручною. Наприклад, для отримання даних використовується функція find у яку можна передавати параметри пошуку. Результатом буде відповідний масив даних, який передається в контролер та відправляється до клієнта. Також можна використовувати такі функції: findOne, findOneById, updateOne, updateOneById, deleteOne, save та інші. Для цього потрібно описати схему класу. Вони зберігаються в папці models. Використовуючи клас Schema, який імпортується із mongoose створюється відповідна схема. В конструктор передається набір полів, які складаються з наступних параметрів: type – тип поля; required: булеве значення, яке відовідає за те, чи є поле обов'язковим; unique – значення, яке задає те, чи має поле бути унікальним. Виклики контролерів здійснюються у функціях, які містяться в папці routes. Файл index.js є основною точкою в програмі. Для полегшення розробки та її зручності використовується фреймворк Express. Створюється зміна app, щоб підключити необхідні функції використовується app.use. Сюди передаються наші файли із папки routes із підключеними контролерами. Перед запуском сервера відбувається підключення до бази даних. Якщо воно пройшло успішно сервер запускається на вказаному вище порті. В даній роботі сервер розміщується локально на порті 5000.

РОЗДІЛ 3 КЛІЄНТСЬКА ЧАСТИНА ПРОЄКТУ

Основним інструментом розробки клієнтської частини стала бібліотека React. Вона дозволяє швидко та зручно будувати односторінкові застосунки (SPA). Перевагою таких веб-проєктів є те, що для користувача вони виглядають як звичайний завантажений додаток, оскільки сторінки не перезавантажуються, а лише оновлюють свій вміст. Це значно покращує користувацький досвід використання.

3.1 Організація роботи з даними.

Робота з даними організована за допомогою бібліотеки Redux Toolkit, яка реалізує архітектуру FLUX. [9] У цій архітектурі можна виділити наступні складові:

- Actions – це функції, які містять назву дії, та payload (деякі дані, які використовуються для оновлення значення)
- Store – основне місце, в якому зберігається стан додатку
- Dispatcher – функція, яка приймає action та вносить відповідні зміни в store
- Views – відображення даних

В загальному цю архітектуру можна описати так. Є одне місце в якому зберігаються всі дані. Для їх зміни потрібно описати функції action. Далі в будь-якому місці програми може бути викликана функція dispatch в яку передається action. Змінені дані записуються до store. Доступ до даних можливий у кожній частині програми. Діаграма FLUX архітектури показано на рисунку 4.

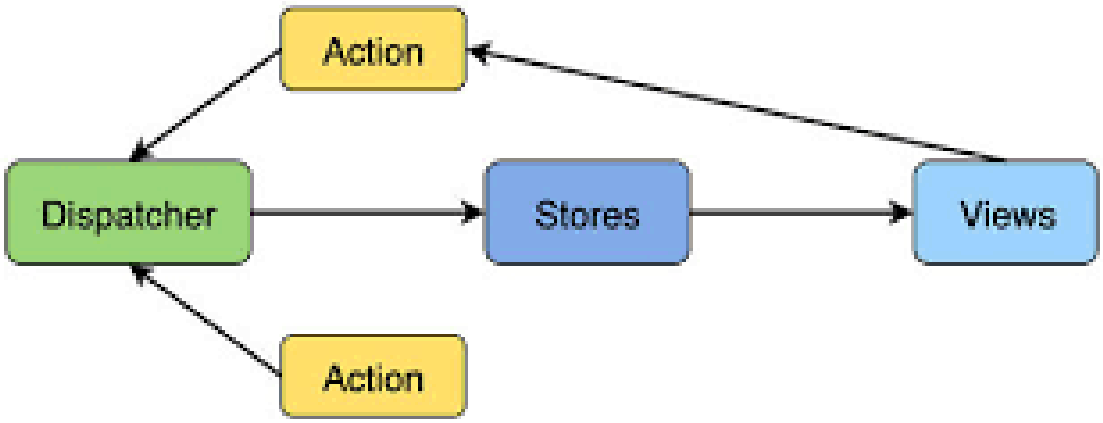

Рисунок 4 - FLUX архітектура

Запити до бази організовані за допомогою Redux Toolkit Query. Цей інструмент надає досить потужне рішення для взаємодії клієнта з сервером. Використовується функція createApi. Параметрами ця функція приймає об’єкт baseQuery, тут задаються певні початкові значення. У даній роботі туди було передано базове посилання на сервер та заголовок авторизації, який далі буде використовуватись для кожного запиту. Ще одним об’єктом, який приймається цією функцією є endpoints. Тут описуються всі запити. При першому виконанні запиту результат записується в store. Після чого, якщо цей запит викликається із тими самими параметрами повертається попереднє значення. Повторний запит відправляється лише при змінні вхідних даних. Це значно пришвидшує роботу застосунку. Використовуються два види запитів query і mutation. Перший використовується для отримання даних, другий для їх зміни. В запит на зміну даних можна передати функцію onQueryStarted, яка буде виконуватись під час того, як відправляється запит. Ця функція приймає параметром dispatch. Таким чином можна внести зміни в store ще до того як запит виконався. Для користувача це виглядає так, ніби програма виконує дії моментально. Така можливість є дуже зручної оскільки запити на отримання даних кешуються. Наприклад, без використання цієї функції після створення нової сутності або її видалення користувач не побачить змін, оскільки вхідні параметри не змінились. Ці функції можна використовувати в будь-якому місці програми. Вони

повертають результат виконання запиту, а також додаткові дані, які вказують чи виконується запит прямо зараз, помилки та інші. Завдяки тому, що при розробці використовувався TypeScript значно підвищилася швидкість розробки. Для кожного запиту було описано типи для вхідних параметрів та результатів. Після цього редактор коду сам вказував, які поля доступні для використання. Також це допомагає відловити помилки ще на етапі розробки.

Проте не всі дані повинні записуватись у store. Для того, щоб створити локальну зміну стану використовується функція useState. В неї передається початкове значення та повертається сама зміна і функція для її оновлення. Приклад такого використання є значення, яке вводить користувач. Воно не потрібно у всій програмі, а лише у тій частині де воно вводиться. Для того, щоб виконувати певні дії залежно від зміни стану використовується функція useEffect. У неї передається callback функція яка повинна бути виконана і масив змінних. При кожному оновленні змінні порівнюються із попереднім значенням і лише, якщо вони відрізняється виконується дія.

3.2 Відображення даних.

Для відображення даних створюються функціональні компоненти, які викликають функції для отримання даних та показують їх користувачеві. Кожна така функція повертає JSX елементи. Раніше для цього в React використовувались класові компоненти, проте зараз такий стиль написання коду можна зустріти дуже рідко.

У даній роботі для розробки візуальної частини клієнтської частини використовувалась бібліотека Material UI. Вона надає багато готових рішень, які часто використовуються, а також великий набір іконок. Для того, щоб змінити стилі компонентів використовується функція createTheme. В неї можна передати основні кольори, які будуть використовуватись, шрифти та інше. Для побудови

власного компоненту із стилями використовувалась функція styled, яка імпортується із Material UI. Туди передається назва HTML тегу, який стилізується та відповідні параметри. Такий метод запису називається CSS in JS. Тобто властивості із мови стилів CSS записуються прямо у JS або TS файли.

Хорошою практикою вважається розбивати відображення на відповідні компоненти, а не тримати все в одному файлі. Це підвищує якість та швидкість розробки, оскільки тоді можна повторно використовувати вже розроблені елементи і не повторюватись. Так було розроблено багато компонентів для представлення різних даних, після чого вони використовувались як будівельні блоки для розробки комплексних відображень.

3.3 Структура проєкту

Проєкт було побудовано утилітою create-react-app. Вона виконує початкову настройку проєкту. Створюються всі необхідні для запуску елементи та встановлюються необхідні модулі. Щоб використовувати TypeScript при створенні потрібно вказати параметр –template typescript. Після чого додаються всі необхідні елементи. Проєкт складається із наступних основних складових:

- Папка api – у ній реалізована вся логіка взаємодії клієнта та сервера
- Папка components – містить в собі всі необхідні візуальні компоненти
- Папка constants – файли із константними значеннями
- Папка pages – у ній зібрані всі сторінки застосунку
- Папка router – реалізує перехід між сторінками
- Папка types – тут описані всі необхідні типи
- Папка utils – зберігає додаткові допоміжні функції

На рисунку 5 показано повну структуру проєкту.

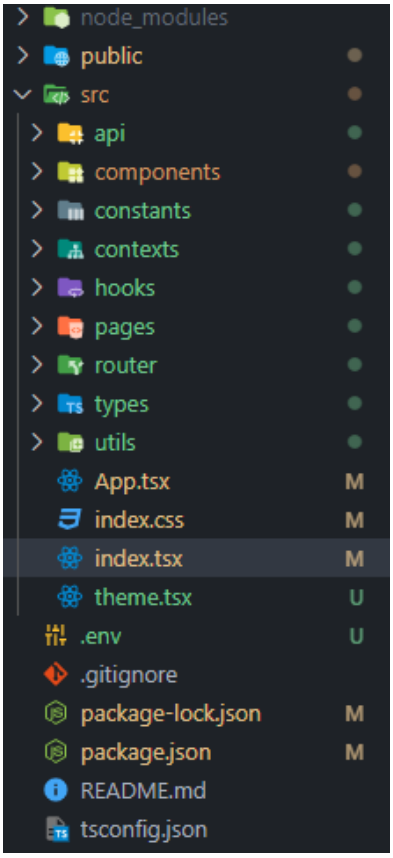

Рисунок 5 - Повна структура проєкту.

Розглянемо деякі особливості реалізації. У папці pages знаходяться три сторінки. Це сторінка входу, реєстрації та головна. Доступ до домашньої сторінки можуть отримати лише авторизовані користувачі. За це відповідає компонент PrivateRoute, який перенаправляє на сторінку входу у випадку відсутності токену або провалу його валідації.

3.4 Реалізація розширення для Google Chrome

Для розробки розширення також використовувалась бібліотека React. Проте організація роботи з даними реалізована набагато простіше ніж в основному проєкті. Оскільки тут потрібно використовувати лише запити на

отримання всіх паролів для заданого посилання та отримання розшифрованого пароля було вирішено не навантажувати застосунок додатковими бібліотеками. Було використано стандартну функцію для здійснення запитів fetch. Для стилізації також не було використано ніяких додаткових інструментів. Для цього використовувалась стандартна мова розмітки CSS.

Для взаємодії із браузером Chrome надається ряд необхідних функцій. Наприклад, для отримання посилання активного вікна потрібно зробити наступне. Визначити об’єкт із полями active: true та lastFocusedWindow: true. Після чого його потрібно передати в функцію chrome.tabs.query. Також вона приймає функцію callback в якій зберігається посилання. Далі воно записується у стан програми і використовується за необхідності.

Щоб протестувати розроблене розширення в браузері Google Chrome, потрібно ввімкнути режим розробника у меню вибору. Після цього потрібно завантажити розроблене розширення. Далі воно буде доступне для використання.

РОЗДІЛ 4 ОГЛЯД ФУНКЦІОНАЛУ ЗАСТОСУНКУ

Основним результатом роботи є готовий до використання web-проєкт для зберігання, створення і відображення паролів та зашифрованих нотаток із розширенням для браузера Google Chrome. Вебсайт складається із сторінок авторизації та домашньої сторінки.

4.1 Двофакторна авторизація

Доступ до основного функціоналу можуть отримати лише користувачі, які пройшли двофакторну авторизацію. Для цього на сторінці входу потрібно ввести вказане при реєстрації ім’я або мобільний номер телефону та пароль. Якщо ж користувач до цього не використовував сайт, йому потрібно створити аккаунт. Форма для входу користувача показана на рисунку 6. Форма створення аккаунта має аналогічний вигляд, але обов’язковими для введення є як ім’я так, і номер телефону.

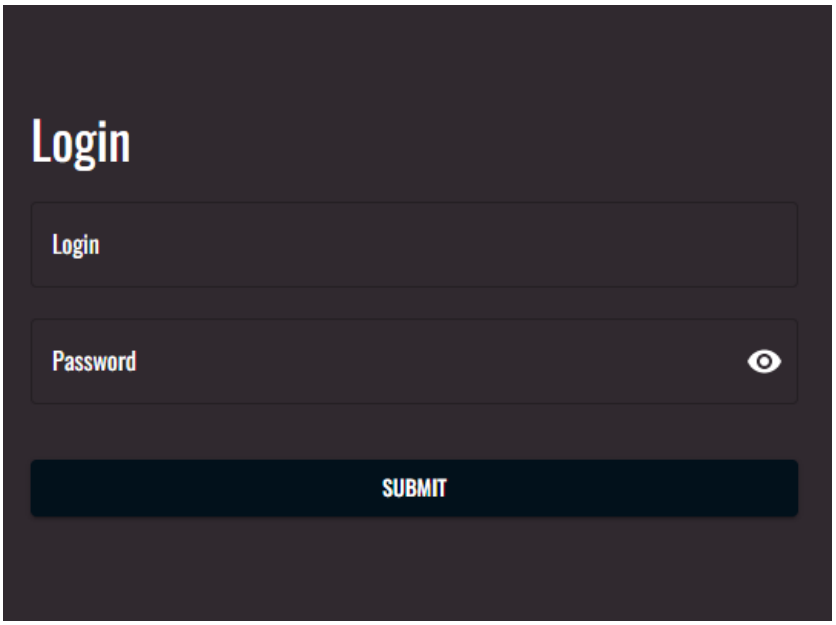

Рисунок 6 - Форма входу

Після проходження основного етапу авторизації користувачеві відправляється 6-ти значний код. Також у тексті повідомлення є назва сервісу, яка вказується при його створенні. Приклад SMS, який отримує користувач показано на рисунку 7.

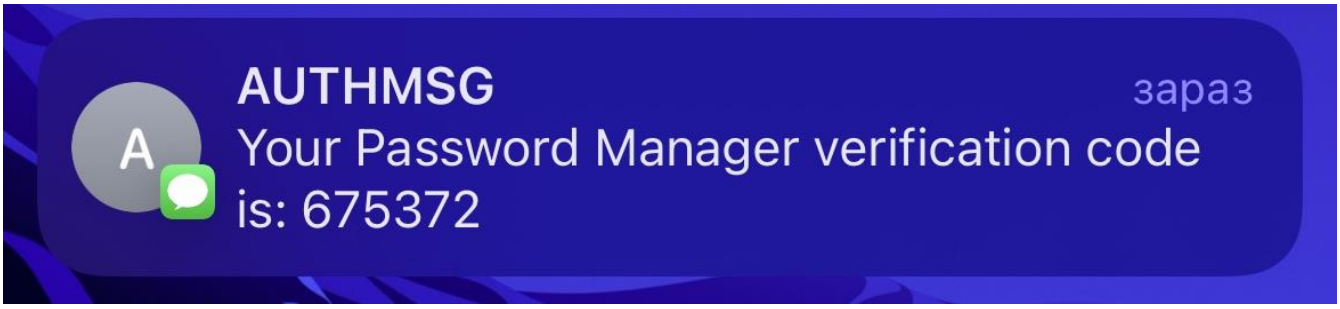

Рисунок 7 - Приклад sms

Після успішного введення цього коду у localStorage записується токен, який надалі буде використовуватись. Користувач після цього перенаправляється на головну сторінку.

4.2 Головна сторінка

Після успішної авторизації користувач потрапляє на головну сторінку. Вона має таку структуру:

- Ліве меню, в якому є можливість вибору паролів або нотаток та виходу з аккаунта
- Ліва частина, в якій відображаються всі створенні паролі або нотатки, рядок пошуку та кнопка створення
- Права частина, яка змінюється залежно від того, що зараз вибрано

Вигляд головного екрану показано на рисунку 8.

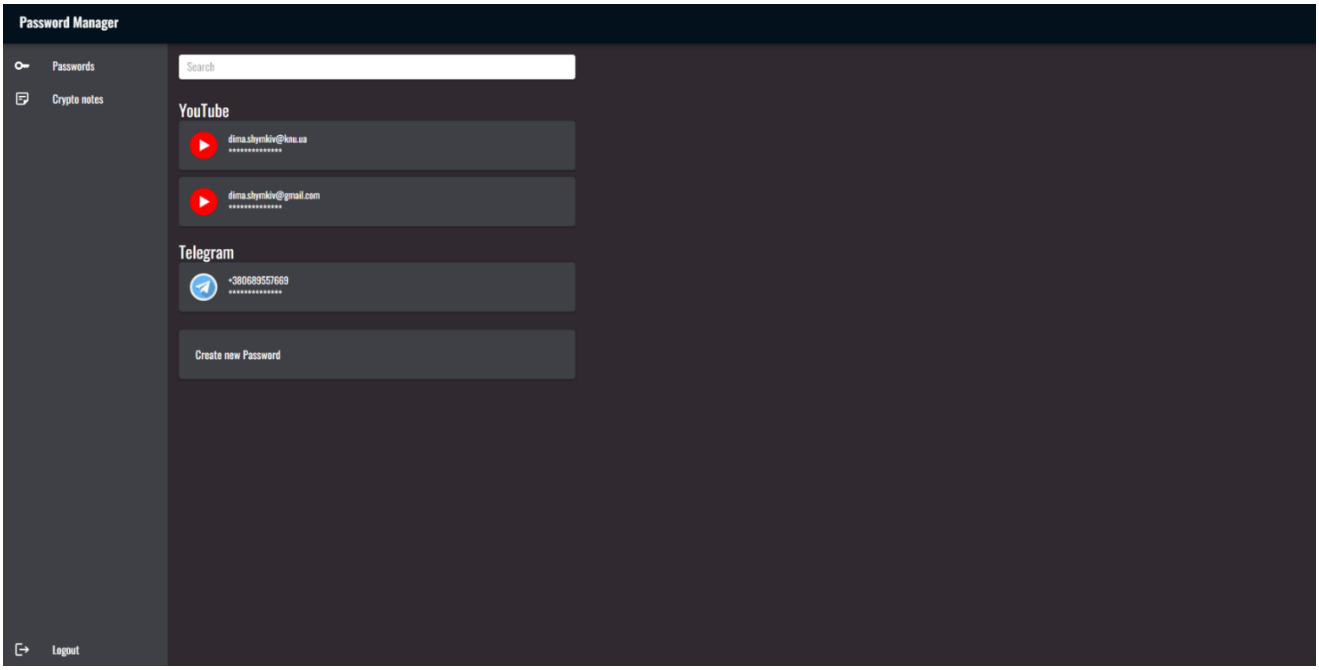

Рисунок 8 - Головний екран

4.3 Робота з паролями

Для того, щоб подивитись інформацію про відповідний пароль, користувачеві потрібно натиснути на нього. Після цього у правій частині програми зміниться вміст та він побачить наступну інформацію. Заданий при створенні логін із можливістю скопіювати його, схований пароль, який можна скопіювати або переглянути. Після успішного копіювання буде показане відповідне повідомлення. Також одним з полів є посилання, якщо воно вказане для додатку. Натиснувши на іконку збоку відкриється нове вікно за заданим посиланням. На рисунку 9 показано, як це виглядає.

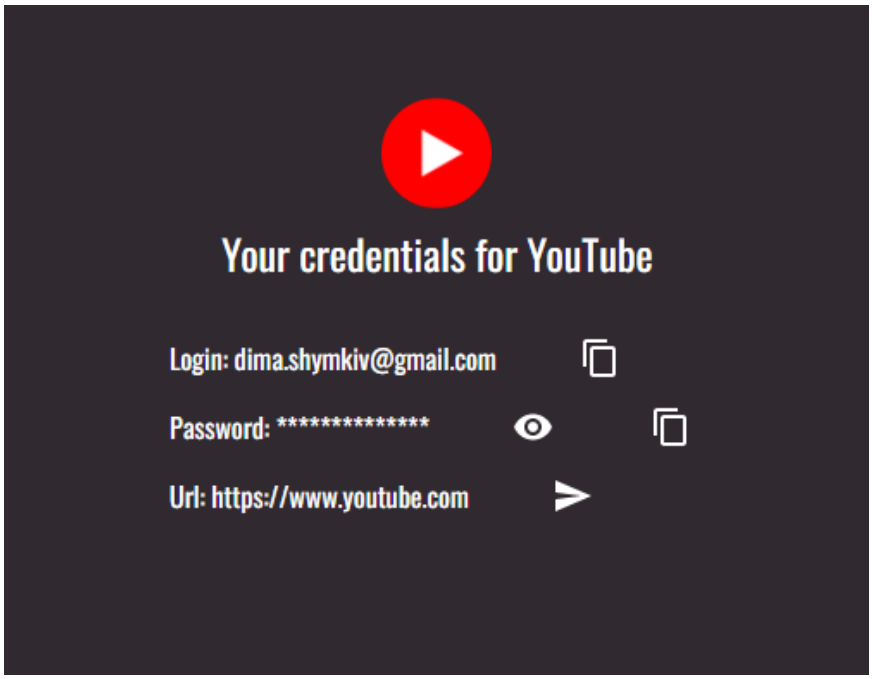

Рисунок 9 - Перегляд пароля

Для того, щоб додати новий пароль користувач повинен натиснути на кнопку “Create new Password”. Для створення нового паролю потрібно вказати логін, пароль та вибрати із випадаючого списку додаток, для якого створюється пароль. Всі поля є обов’язковими. На рисунку 10 показано вигляд форми для створення паролю.

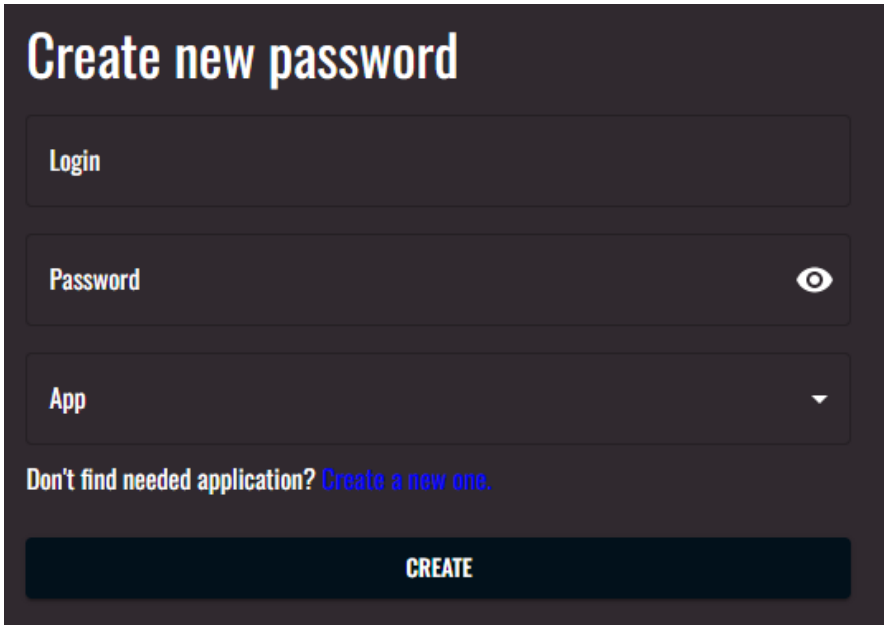

Рисунок 10 - Форма створення пароля

Якщо користувач не знайде потрібного додатку у списку, він може створити власний. Для цього йому необхідно вказати ім’я додатку. Якщо це вебсайт, користувач може вказати посилання для зручності використання паролів. Також користувач може завантажити логотип додатку, проте це не є обов’язковим. Форма створення додатку показана на рисунку 11.

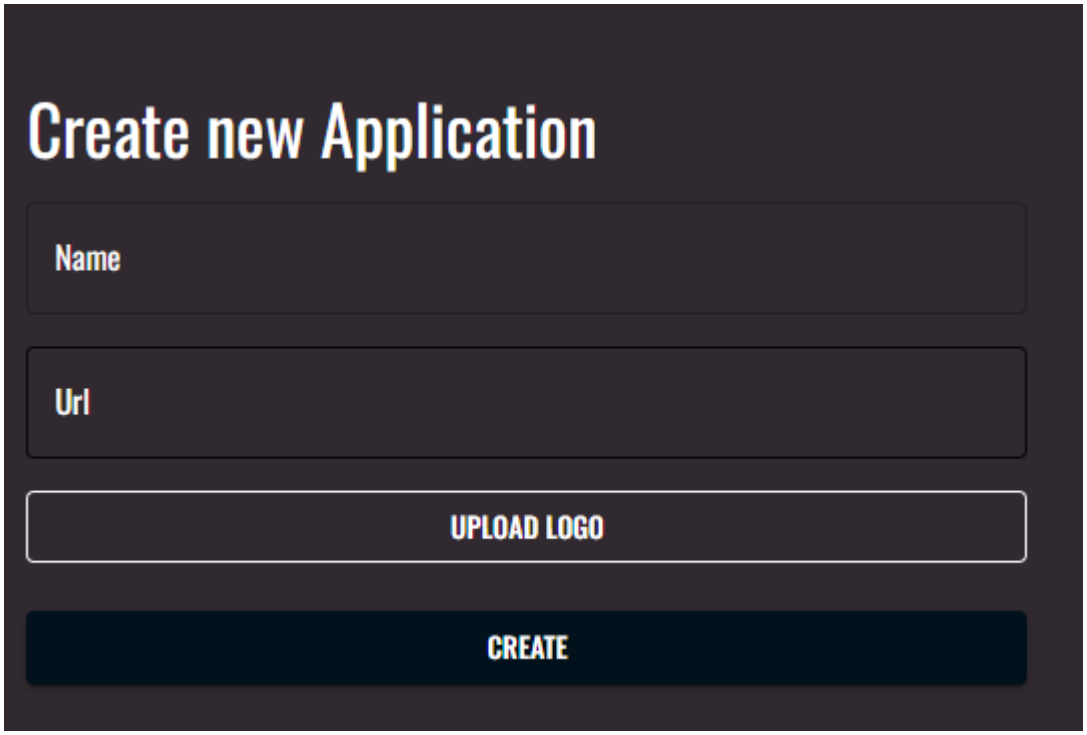

Рисунок 11 - Форма створення додатку

4.4 Шифровані нотатки

Натиснувши у лівому меню кнопку “Crypto notes”, ліва частина програми змінить свій вміст і буде показувати усі нотатки, які створив користувач. Цей список схожий на той, який відображається для паролів. Він також має можливість фільтрації даних. Для цього у полі пошуку потрібно почати вписувати текст. Список автоматично буде фільтруватись та показувати лише ті нотатки, у яких в назві міститься заданий текст. Також тут присутня кнопка створення нової нотатки.

Вигляд списку нотаток можна побачити на рисунку 12.

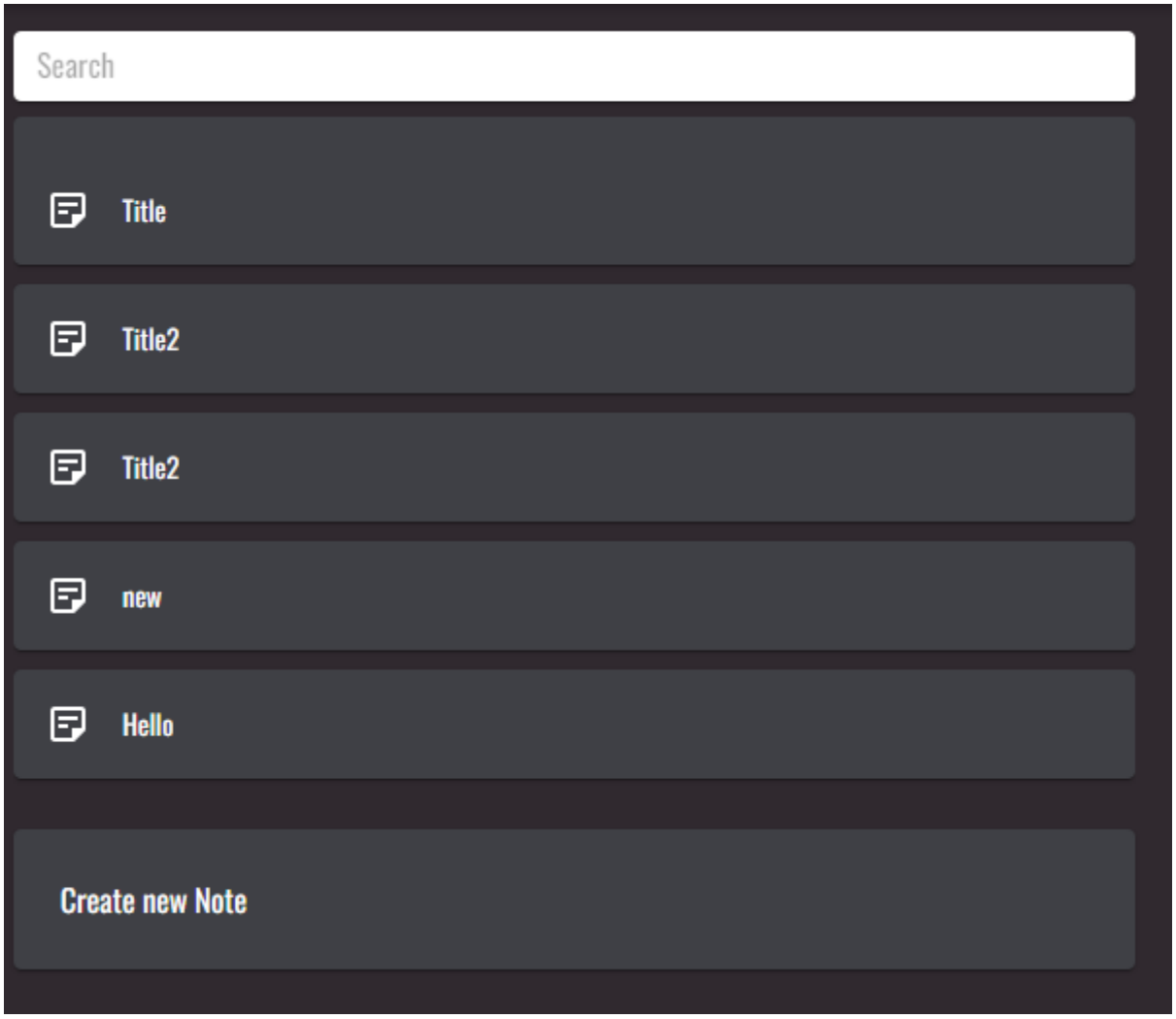

Рисунок 12 - Вигляд списку нотаток.

Для створення нової нотатки користувач повинен натиснути на кнопку “Create new Note”. Після цього у правій частині програми з’явиться форма створення. Тут потрібно вказати назву нотатки та секретний ключ на основі якого буде проводитись шифрування. Цей ключ потрібно запам’ятати, оскільки далі він використовується для отримання вмісту нотатки. Також користувач побачить тут текстовий редактор. В ньому є можливість створювати заголовки різних розмірів, додавати жирний, курсивний та підкреслений текст. Також вставляти нумеровані та ненумеровані списки.

На рисунку 13 продемонстровано, як виглядає форма створення нотатки.

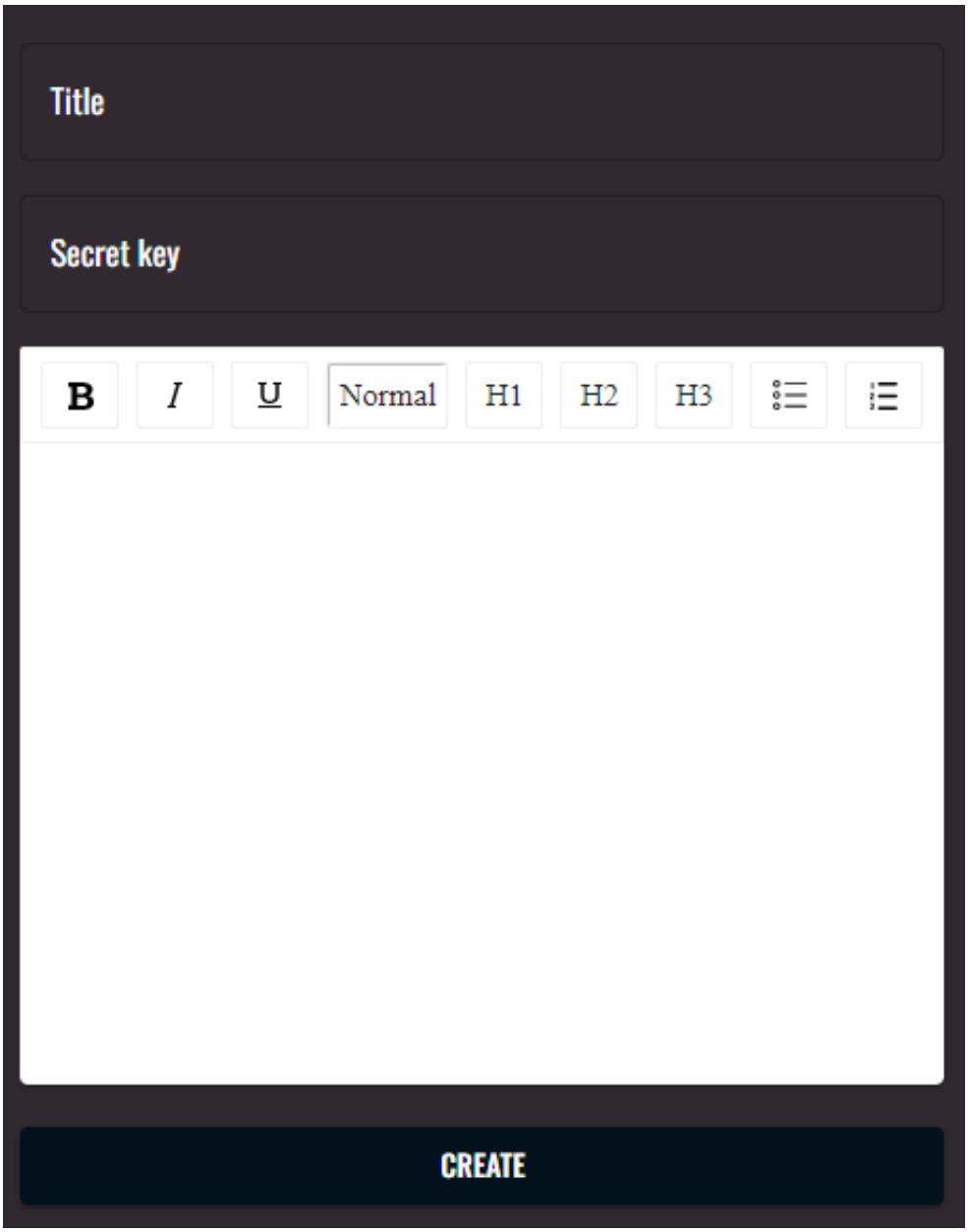

Рисунок 13 - Форма створення нотатки

Натиснувши на будь-яку створену нотатку, користувач у правому вікні побачить її назву та поле для вводу. В це поле потрібно ввести секретний ключ, який користувач вигадував при створенні. За його допомогою буде проводитись розшифрування нотатки. Якщо секретний ключ введено невірно користувач не зможе побачити вміст нотатки, оскільки процес дешифрування не буде проведено успішно.

Поле вводу ключа показано на рисунку 14.

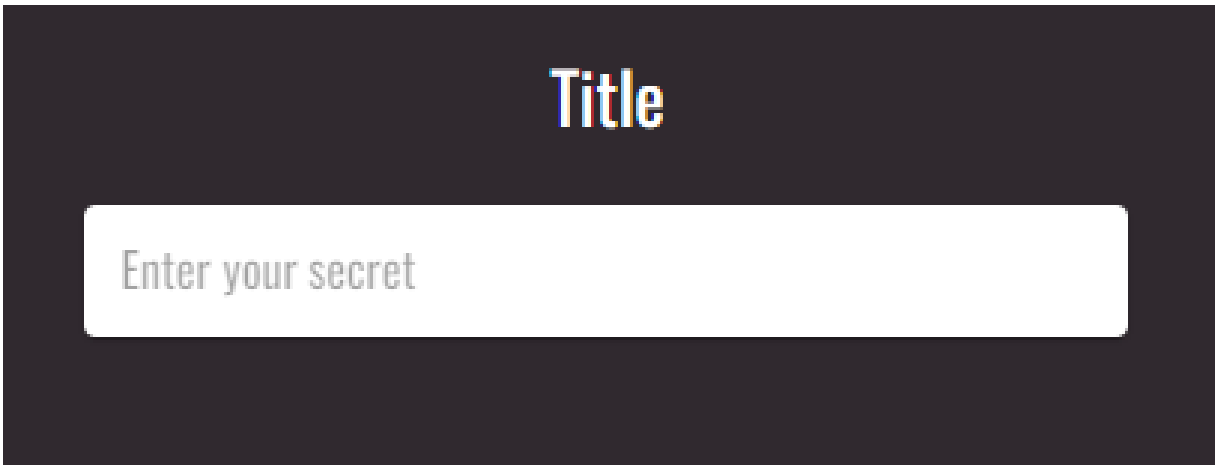

Рисунок 14 - Поле вводу ключа

Після успішного введення секретного ключа поле вводу заміниться на справжній текст нотатки. Тоді користувач зможе переглядати створений раніше текст. Приклад нотатки показано на рисунку 15.

Рисунок 15 – Текст нотатки

4.5 Розширення для Google Chrome

Розширення для браузера Google Chrome використовується, щоб отримати більш зручний спосіб використовувати збереженні паролі. Перейшовши за посиланням, яке було вказано при створенні додатку, розширення відправить на сервер запит для отримання всіх створених паролів для цього додатку. Тоді буде показано список із цих паролів. Користувач матиме можливість скопіювати необхідний пароль та вставити його у відповідне поле. Вигляд розширення показано на рисунку 16.

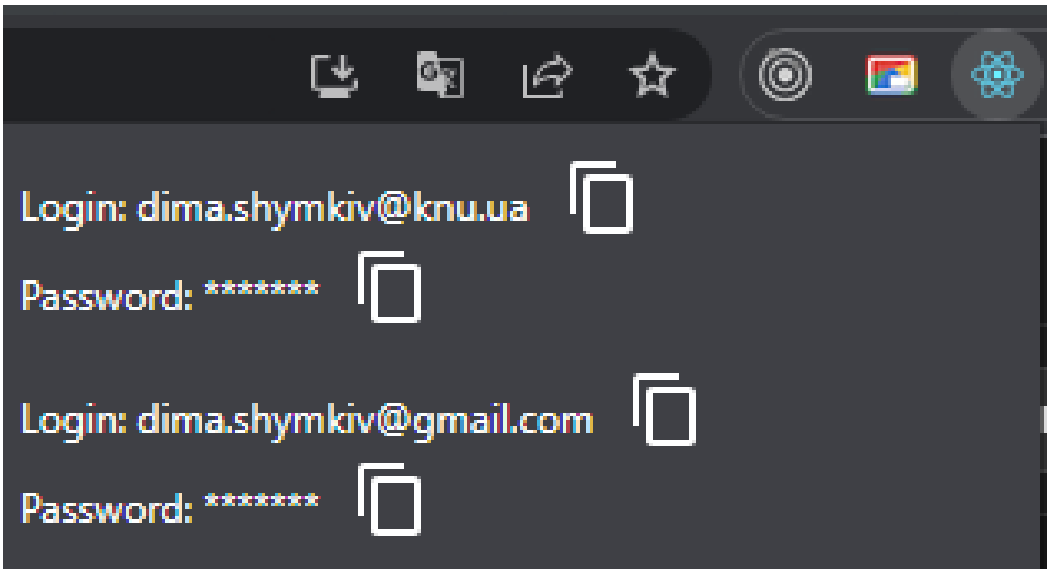

Рисунок 16 - Розширення для Google Chrome

ВИСНОВКИ

Під час виконання роботи були виконані всі поставлені задачі. А саме:

- Проведено аналіз технологій та інструментів, які використовуються при створенні web-проєктів та вибрано найбільш зручні із них
- Розроблена система двофакторної авторизації за допомогою паролю та коду надісланого у повідомленні на номер мобільного телефону
- Спроектована схема збереження даних у БД
- Розроблено серверну частини та побудовано REST API для обміну даними між клієнтом та сервером
- Розроблено користувацький інтерфейс
- Забезпечено надійність зберігання даних
- Отримано новий досвід у розробці розширень для браузерів. А саме розширення для Google Chrome.
- Поглиблено знання у створенні web-застосунків із використанням мов програмування JavaScript та TypeScript. А також бібліотеки React і фреймворку Express.

Результатом виконання практичної частини роботи є web-проєкт із клієнтською та серверною частиною. Розроблений продукт є логічно завершеним.

ДЖЕРЕЛА

1. Каскадна модель програмування [Електронний ресурс] - Режим доступу до ресурсу : https://evergreens.com.ua/ua/articles/software-development-metodologies.html

2. Бібліотека express [Електронний ресурс] - Режим доступу до ресурсу : https://expressjs.com/

3. Інформація про NoSql БД MongoDB [Електронний ресурс] - Режим доступу до ресурсу : https://uk.wikipedia.org/wiki/MongoDB

4. Бібліотека Mongoose [Електронний ресурс] - Режим доступу до ресурсу : https://mongoosejs.com/

5. Інформація про JWT [Електронний ресурс] - Режим доступу до ресурсу : https://uk.wikipedia.org/wiki/JSON_Web_Token

6. Бібліотека Draft.js [Електронний ресурс] - Режим доступу до ресурсу : https://draftjs.org/

7. Нереляційні БД [Електронний ресурс] - Режим доступу до ресурсу: https://www.mongodb.com/nosql-explained

8. Алгоритм шифрування AES [Електронний ресурс] - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Advanced_Encryption_Standard

9. FLUX архітектура [Електронний ресурс] - Режим доступу до ресурсу : https://www.freecodecamp.org/news/an-introduction-to-the-flux-architectural-pattern-674ea74775c9/

ДОДАТОК А ПРИКЛАД РЕАЛІЗАЦІЇ КОНТРОЛЕРА

```
const UserService = require("../services/User.service");
const { verifyPassword } = require("../utils/Password.utils");
const { generateToken } = require("../utils/Token.utils");

class UserController {
  async create(req, res) {
    const body = req.body;
    const validation = await UserService.validateUser(body);
    if (!validation.isValid) return res.status(400).send(validation.error);

    await UserService.create(body);
    return res.status(201).send("Success");
  }

  async login(req, res) {
    const { userLogin, masterPassword } = req.body;

    const user = await UserService.findUser(userLogin);
    if (!user)
      return res
        .status(400)
        .send("Can't find user with that email or phone number");

    const isPasswordRight = await verifyPassword(
      masterPassword,
      user.masterPassword
    );

    if (isPasswordRight) res.status(200).json(user);
    else res.status(400).send("Invalid password");
  }
```

```
  async auth(req, res) {
    const user = await UserService.finUserById(req.user.id);
    if (!user) return res.status(400).send("User not found");
    const token = generateToken({ id: user._id });
    return res.status(200).json({ user: user._doc, token });
  }
}

module.exports = new UserController();
```

ДОДАТОК Б ПРИКЛАД РЕАЛІЗАЦІЇ СЕРВІСА

```
const User = require("../models/User.model");
const { generateHashedPassword } = require("../utils/Password.utils");

class UserService {
  async create(body) {
    const { masterPassword } = body;

    const hashedPassword = await generateHashedPassword(masterPassword);
    const user = new User({
      ...body,
      masterPassword: hashedPassword,
    });

    await user.save();
    return user;
  }

  async validateUser(body) {
    const { username, phoneNumber, masterPassword } = body;

    if (!username  !phoneNumber  !masterPassword)
      return { isValid: false, error: "All fields should be filled" };

    const isUserAlreadyExist = await this.checkSameUser({
      username,
      phoneNumber,
    });

    if (isUserAlreadyExist)
      return { isValid: false, error: isUserAlreadyExist };

    return { isValid: true };
```

```
  }

  async checkSameUser(user) {
    for (const key in user) {
      const usersWithSameValue = await User.find({ [key]: user[key] });
      if (usersWithSameValue.length)
        return User with this ${key} already exist;
    }
    return "";
  }

  async findUser(userLogin) {
    const user = await User.findOne({
      $or: [{ username: userLogin }, { phoneNumber: userLogin }],
    });
    return user;
  }

  async finUserById(id){
    const user = await User.findOne({ _id: id });
    return user;
  }
}

module.exports = new UserService();
```