МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему «Програмний додаток генерації та збереження паролів "Safe Password Storage"»

за спеціальністю 122 «Комп'ютерні науки»
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ-71 Толстоноженко Станіслав Олегович

Кваліфікаційна робота бакалавра
захищена на засіданні ЕК
з оцінкою **_____ «____»____2021р.**

Науковий керівник: _____ к.т.н., доц. Шендрик В.В
(підпис)

Голова комісії _____ Шифрін Д.М._____
(підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань

Студент _____
(підпис)

Суми-2021

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук
Секція інформаційних технологій проектування
Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»

ЗАТВЕРДЖУЮ

Зав. секції ІТП

_____В.В. Шендрик

«___»_____2020 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Толстоноженко Станіслав Олегович

1 Тема роботи *Програмний додаток генерації та збереження паролів "Safe Password Storage"*

керівник роботи *Шендрик Віра Вікторівна, к.т.н., доцент*,

затверджені наказом по університету від *« 14» квітня 2021 р. № 0181-VI*

2 Строк подання студентом роботи *« 07 » червня 2021р.*

3 Вхідні дані до роботи *технічне завдання на розробку програмного додатку менеджера паролів*

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *аналіз предметної області, проектування програмного додатку, розробка програмного додатку*

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

6. Консультанти розділів роботи:

Розділ	**Консультант**	**Підпис, дата**	
		Завдання видав	**Завдання прийняв**

7.Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Інціалазація проекту	18.01.21-20.01.21	
3	Затвердження теми	21.01.21-21.01.21	
4	Планування проекту	22.01.21-09.02.21	
5	Розробка клієнтського сервера	10.02.21-18.03.21	
6	Розробка файлового сервера	19.03.21-06.05.21	
7	Тестування проекту	07.05.21-12.05.21	
8	Оформлення результатів виконання	13.05.21-18.05.21	
9	Створення документації	19.05.21-24.05.21	
10	Інсталяція проекту на хостинг	25.06.21-08.06.21	

Студент _____ Толстоноженко С.О.

Керівник роботи _____ к.т.н., доц. Шендрик В.В

РЕФЕРАТ

Тема кваліфікаційної роботи «Програмний додаток генерації та збереження паролів "Safe Password Storage"»

Дана кваліфікаційна робота бакалавра складається зі вступу, трьох основних розділі та висновку.

У першому розділі проведено аналіз актуальності досліджуваної задачі, проаналізована робота програмних продуктів-аналогів, розроблена мета та опрацьовані задачі майбутнього проекту.

У другому розділі розроблено моделювання програмного додатку менеджер паролів за допомогою IDEF моделі, створено необхідні UML-моделі та проведено моделювання бази даних.

Третій розділ містить детальний опис підготовки до початку роботи з майбутнім проектом, процес створення програмного забезпечення з описом всіх необхідних файлів та алгоритмів роботи проекту, а також покроковий опис використання користувачем програмного додатку.

Кінцевим результатом даної роботи є створення web-додатоку генерації та збереження паролів за допомогою якого користувач матиме змогу знаючи один звичайний пароль генерувати унікальні паролі для входу в будь-які онлайн сервіси.

Пояснювальна записка до роботи містить 141 сторінок, 79 рисунків, 16 таблиць, 3 додатків та 11 джерел.

Ключові слова: Javascript, web-додаток, Angular, MongoDB, Node.js, менеджері паролів, MySQL, TypeScript.

ЗМІСТ

ВСТУП

Паролі досить часто критикують як з точки зору безпеки, так і зручності, що робить їх менш ідеальним методом аутентифікації. Водночас паролі як ключі до онлайн-профілів вважаються першим і часто єдиним способом захисту цифрових даних користувачів. Більшість кібератак відбувається за для викрадення паролів з метою отримання конфіденційних даних.

Крадіжка даних може бути тісно пов’язана з фішингом чи іншими методами соціальної інженерії, наприклад, як спам, що являє собою масове розсилання небажаних листів найчастіше за допомогою електронної пошти з метою поширення шкідливих вкладень або посилань.

Для захисту особистої інформації користувачам доводиться використовувати складні паролі, що призводить до складності їх запам’ятовування при використанні на різних сайтах. Отже, виникає питання про способи зберігання паролів. Одним з найпростіших способів є запис паролю на папірець. Однак його легко втратити чи він може бути викраденим. У зв’язку з цим більш ефективним способом зберігання паролів є менеджер паролів.

Менеджер паролів – це програмне забезпечення, яке допоможе користувачу працювати з паролями та PIN-кодами. За допомогою вказаної програми є можливість генерувати унікальні, а головне надійні паролі, та зберігати їх в програмі.

Основною метою даного проекту є розробка програмного забезпечення для захисту особистих інтернет-даних користувачів шляхом генерації та збереження унікальних паролів. Пріорітетною задачею є розробка web-додатку, що має бути зручним та простим у використанні. Для користувачів буде реалізовано створення хешів, за допомогою яких будуть створюватися унікальні паролі, тому сам пароль запам’ятовувати не потрібно.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність досліджуваної задачі

З розвитком сучасних інформаційних технологій майже кожна людина користується комп’ютером чи ноутбуком. Все частіше люди спілкуються за допомогою соціальних мереж, черпають інформацію з інтернет-сайтів, проводять онлайн розрахунки та інше. У зв’язку з цим все частіше виникає питання про можливі способи збереження паролів, які є основним ключем до особистої інформації. Одним із варіантів розв'язання вказаної проблеми є створення програми для генерації та збереження паролів.

Використання різних менеджерів паролів допомагає користувачу створювати сотні унікальних і сильних паролів та не запам’ятовувати їх, бо інформація зберігається в базі даних. Але все частіше деяким користувачам хотілося б мати один пароль для всіх сервісів. Основним недоліком втрати цього пароля при зломі одного з сервісів є те, що зловмисник автоматично матиме доступ до всіх інших сервісів.

Отже, виникає необхідність у створенні програмного засобу, за допомогою якого користувач матиме змогу створювати та генерувати захищені паролі, дотримуючись таких правил:

- довжина паролю рекомендовано має складатися не менше як з восьми символів;
- створений пароль має складатися з латинських літер abcd…… та інших;
- пароль має містити латинські літери у великому регістрі;
- пароль має містити цифрові позначення;
- пароль має містити спеціальні символи такі, як @,%,#,! та інші.

Більшість менеджерів паролів у своїй роботі не дотримуються всіх цих правил, зазвичай немає спеціальних символів, цифр або змінного регістра [1].

Навіть якщо згенерувати стійкий пароль у вигляді ключового слова, користувачу важко або неможливо його запам’ятати. Унікальний пароль потрібно зберігати в базі даних, спочатку зашифрувавши, а це може призвести до небажаних наслідків таких, наприклад, як взлом. Ось чому виникла ідея створення програмного продукту, який дозволить зручно, а головне безпечно створювати паролі для різних інтернет сайтів.

1.2 Аналіз програмних продуктів для управління паролями

Під час проведення аналізу програмних продуктів-аналогів, які використовують користувачі під час своєї роботи з паролями, були розглянуті такі їх види, як LastPass, Dashlane та KeePass.

LastPass – це сервіс для керування паролями та їх безпечного зберігання.

Захист даних здійснюється на основі шифрування AES-256. Перевагами цієї програми є її багатоплатформність, вона має безплатну версію, привабливий зовнішній вигляд, інтуїтивно зрозумілий інтерфейс, а також можливість синхронізувати дані [2].

Недоліками програми є необхідність збереження своїх даних на віддаленому сервері. Це означає, що користувач не має контролю над сховищем паролів через що існує ризик можливого злому сервера чи облікового запису з витоком всіх збережених паролів [3]. На рис. 1.1 зображено вигляд вказаного програмного забезпечення.

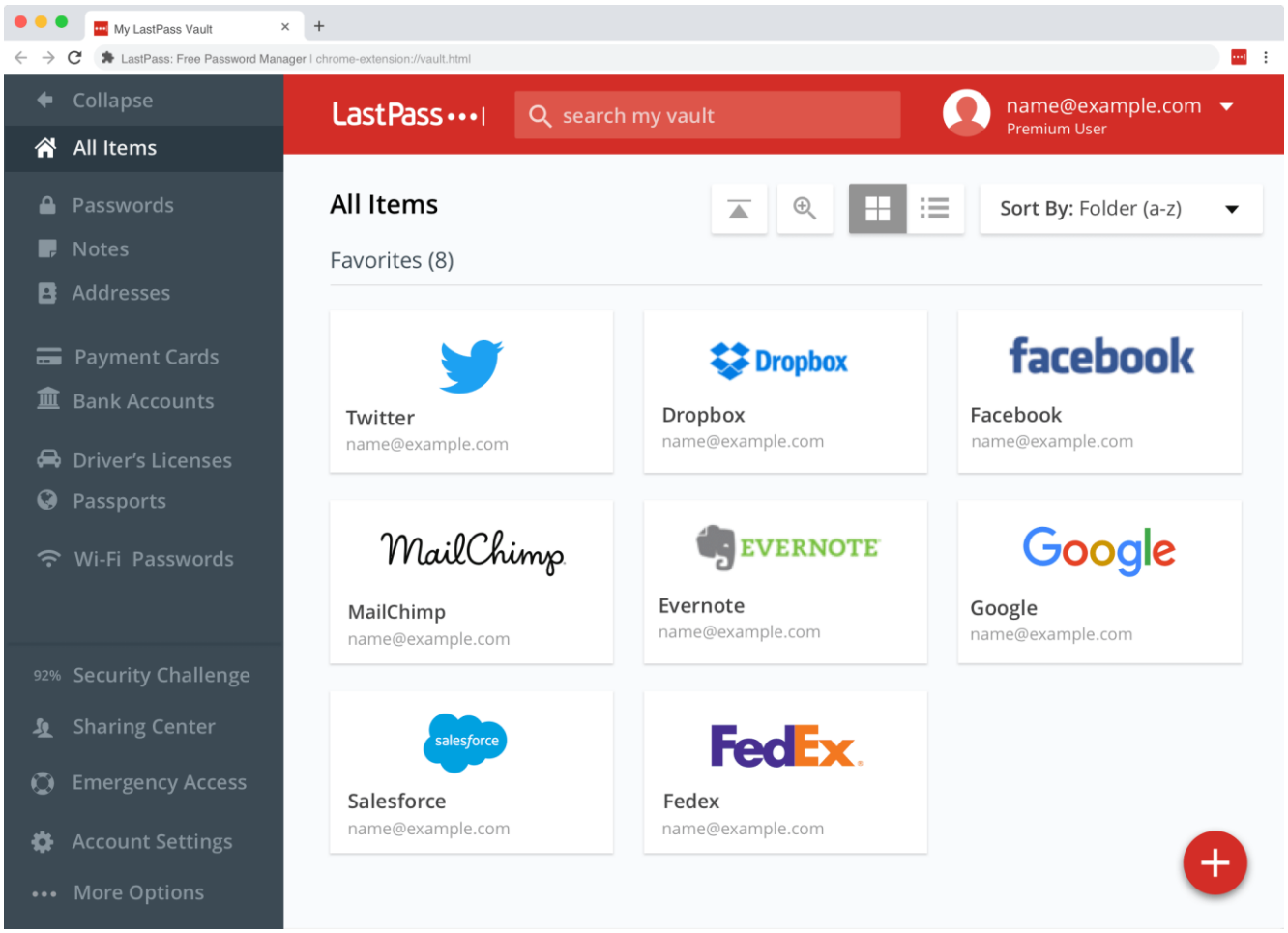


Рисунок 1.1 – Програма LastPass

Dashlane – це програма менеджер паролів, яка має як вебверсію, так і програму для ПК.

У програмі захист даних здійснюється на основі алгоритму шифрування AES-256. В порівнянні з LastPass ця програма має можливість зберігати свою базу даних на особистому пристрої, має двофакторну авторизацію за стандартом 2FA в безкоштовній версії, та U2F в платному тарифі [4].

Недоліками цієї програми є обмежений обсяг зберігання ключів для безкоштовної версії, а саме 50 паролів. Була знайдена вразливість через особливості роботи сервісу з PIN-кодами [3].

На рис. 1.2 зображено вигляд вказаного програмного забезпечення.

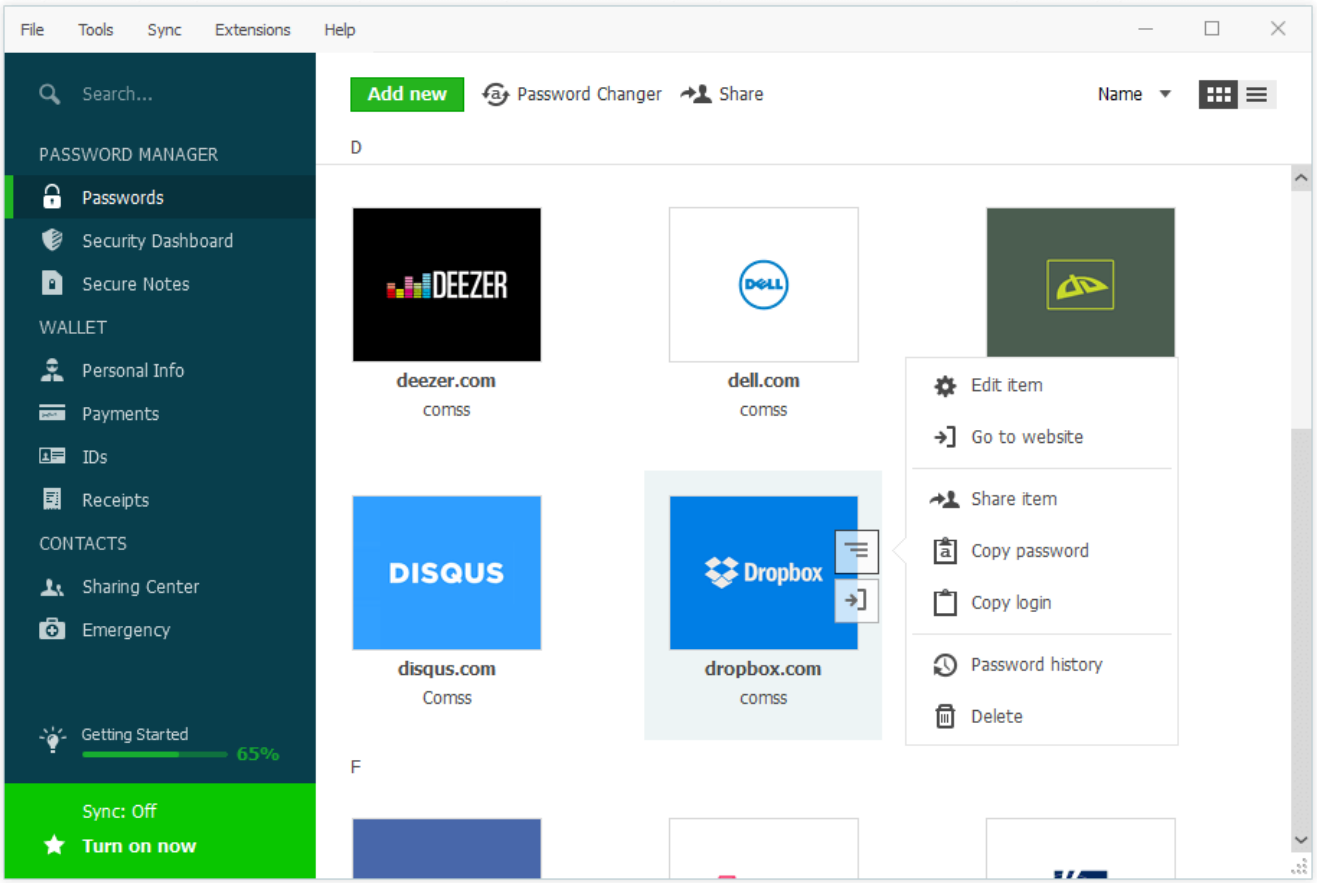


Рисунок 1.2 – Програма Dashlane

KeePass – це безкоштовний менеджер паролів з відкритим кодом.

Захист даних здійснюється на основі алгоритмів шифрування AES-256 і ChaCha20. Плюсом цієї програми є те, що ці алгоритми мають різний принцип роботи. Таким способом можна захистити користувача від спроб прямого дешифрування бази, оскільки хакер не буде точно впевнений в тому, який саме протокол потрібно застосувати для цього [5].

Недоліком цієї програми є MitM-атака це коли програма завантажує оновлення. Під час передачі незашифрованих пакетів є можливість підміни потрібного пакету даних на заражений. У разі успіху зловмисник може відправити запит на експорт всіх ключів, що зберігаються в активних базах [3].

На рис. 1.3 зображено вигляд вказаного програмного забезпечення.

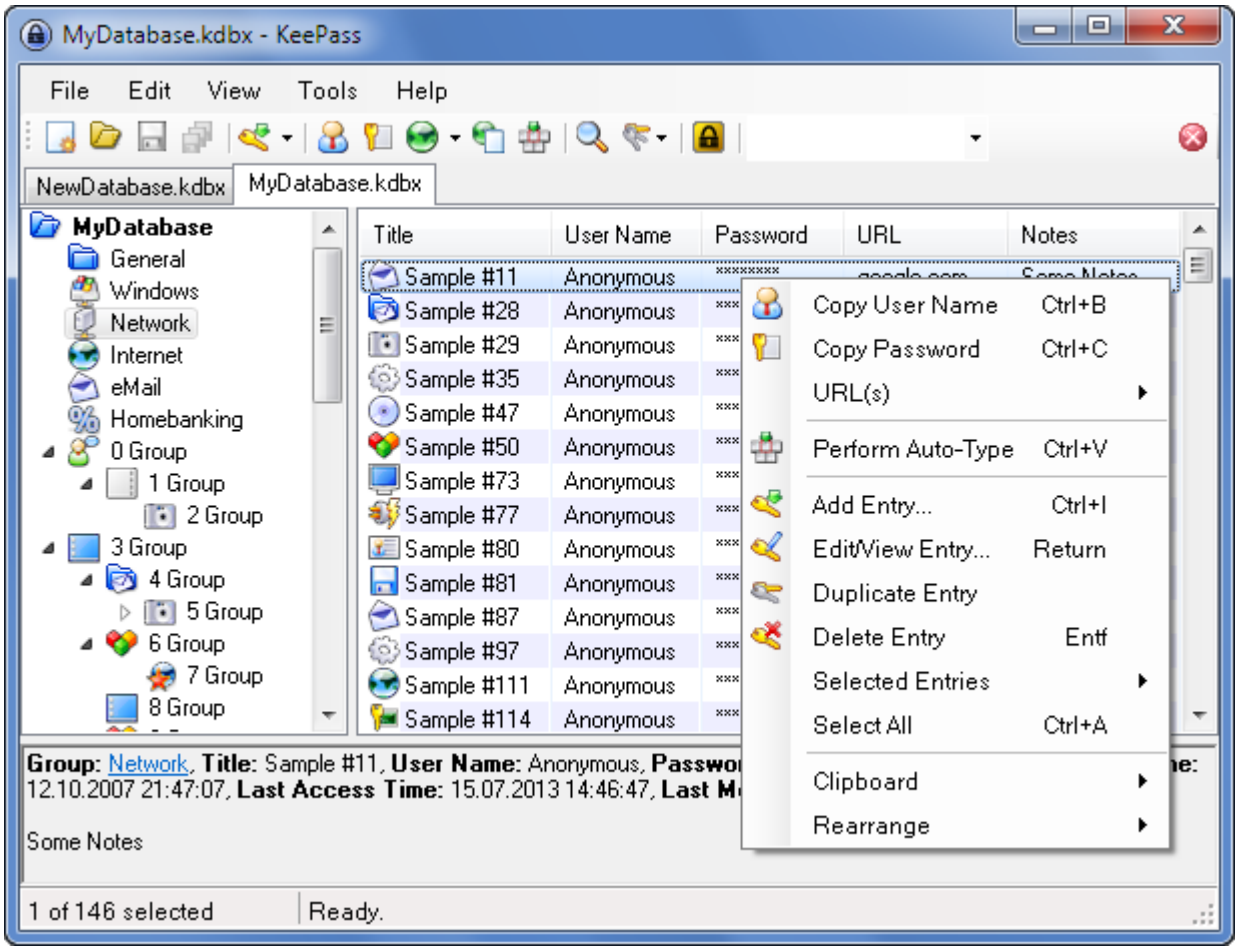


Рисунок 1.3 – Програма KeePass

Проаналізувавши усі аналоги, визначивши їхні переваги й недоліки в зберіганні паролів, було сформовано список задач для власного програмного забезпечення. А саме:

– використовувати клієнт-серверну архітектуру для багатоплатформності;

– на сервері програма повинна зберігати тільки хеш, а не паролі;

– можливість створення необмеженої кількості паролів;

– програма повинна мати декілька різних алгоритмів шифрування, що використовують різний за параметрами хеш, який буде зберігатися в базі даних, яка не потребує опису схеми таблиць;

– для захисту від MITM атак використовувати протокол HTTPS.

В результаті отримаємо продукт, що покриває недоліки наявних аналогів і забезпечує більш надійний захист від злому.

1.3 Постановка задачі

Метою кваліфікаційної роботи є створення програми у вигляді web-додатку, яка допоможе користувачу використовувати один пароль для генерації унікальних паролів для входу в усі сервіси. Під час створення паролю сервер буде отримувати назву сайту і “звичайний” пароль, який ввів користувач, потім сервер створить унікальний пароль, який буде використовуватися для входу на сайт. Програма не зберігає паролі, а тільки створює хеш для кожного сервісу, за допомогою якого генерується пароль для входу.

Для вирішення вказаної мети необхідно виконати наступні задачі:

- Створити два сервери за допомогою фреймворка Node.js, на яких буде реалізована вся логіка web-додатку.
- Створити бази даних для кожного Node.js сервера для зберігання інформації про користувача та хеш.
- Створити сервер за допомогою фреймворка AngularJS, що призначений тільки для зручності взаємодії користувача з web-додатком. На ньому відбувається валідація даних і обробка запитів з сервера Node.js.
- Розмістити проект на хостингу.
- Виконати тестування проекту.

Для реалізації продукту проекту потрібен хостинг. Найкраще підійде провайдер HostPro.ua так як він має найкращу технічну підтримку серед конкурентів. В якості середовища розробки буде використано Visual Studio, шаблоном проекту буде web-програма Node.js. За допомогою нього буде створено два сервери: перший для взаємодії з користувачем - клієнтський сервер з

використанням бази даних MySQL; другий сервер - головний для збереження хеш-кодів. Для його роботи буде реалізовано базу даних mongoDB. MongoDB не вимагає опису схеми таблиць в порівнянні з реляційними БД. Дані зберігаються у вигляді колекцій, у яких не обов'язково повинна бути схожа структура даних, та документів. Оскільки фреймворк AngularJS використовує концепцію Model-View-Controller, його буде використано в якості web-додатку з можливістю оптимізації під різні платформи.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

2.1 Моделювання програмного додатка генерації та збереження паролів "Safe Password Storage" в нотації IDEF0

2.1.1 Функціональне моделювання програмного додатка генерації та збереження паролів "Safe Password Storage" в IDEF0

За допомогою методології IDEF0 розробляється декілька діаграм, з описом функції або процесу:

- контекстна діаграма;
- діаграма верхнього рівня;
- набір дочірніх діаграм, на яких зображено більш детальне уявлення про об'єкт моделювання.

Вхідними даними будуть дані про користувача та звичайний пароль що ввів користувач.

Обмеження складаються з:

- алгоритму створення унікального пароля;
- статистичних даних;
- алгоритму шифрування інформації;

Ресурси програми будуть складатися з двох серверів та двох баз даних для кожного сервера.

Результатом роботи буде унікальний пароль для користувача.

Контекстна діаграма процесу створення пароля зображено на рис. 2.1.

Рисунок 2.1 – Контекстна діаграма процесу створення пароля

2.1.2 Декомпозиція першого рівня контекстної діаграми в нотації IDEF0

Декомпозиція першого рівня контекстної діаграми в нотації IDEF0 має за мету вивчення та аналізу взаємозв’язків організацій, а також змісту та структури.

На діаграмі можна побачити послідовність створення унікального пароля. Спочатку проводиться перевірка вхідних даних про користувача, потім ці дані зберігаються на сервері та робиться запит на головний сервер для створення паролю. На головному сервері відбувається генерація хешу. За допомогою нього і звичайного пароля створюється унікальний пароль з використанням унікального алгоритму [6].

Функціональне моделювання програми генерації та збереження паролів "Safe Password Storage" зображено на рис. 2.2.

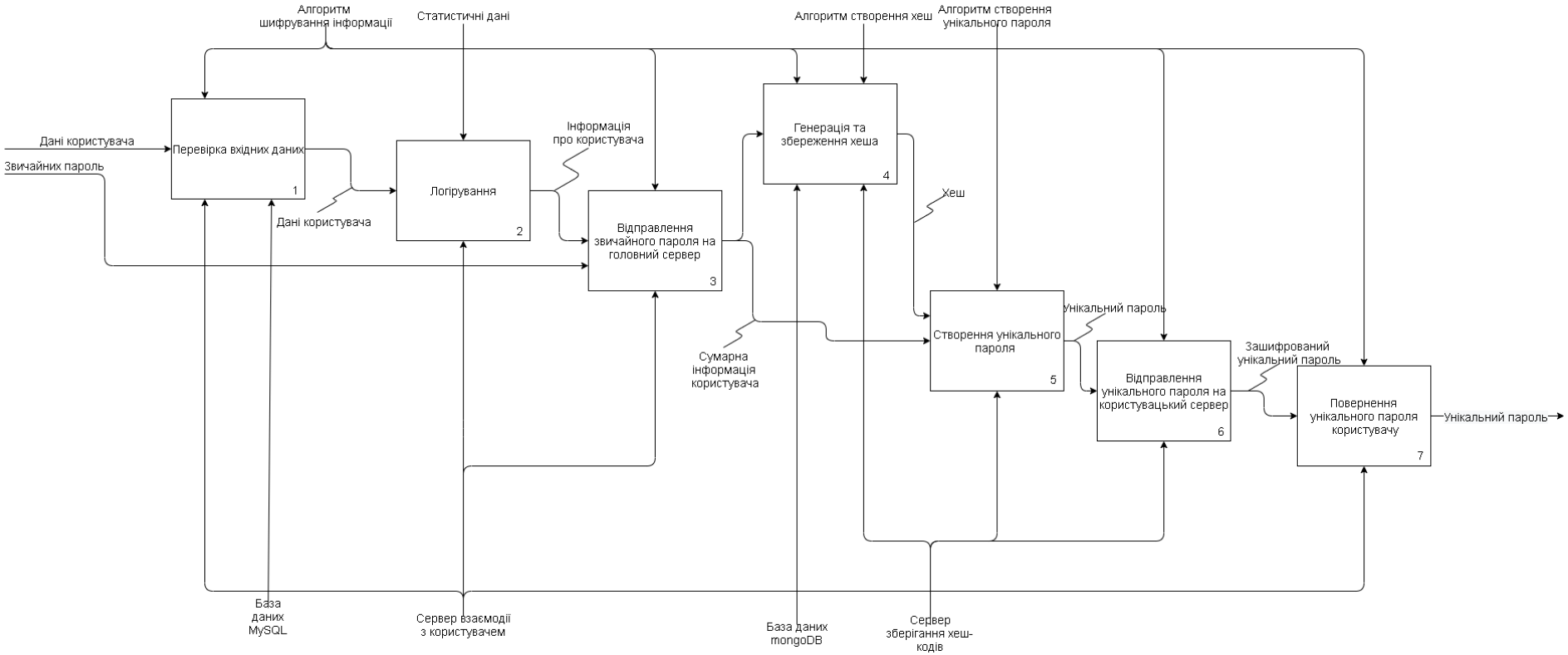


Рисунок 2.2 – Діаграма декомпозиції процесу створення пароля

2.1.3. Структурне моделювання програмного додатку генерації та збереження паролів "Safe Password Storage"

Діаграма станів автомата необхідна для того, щоб показати поведінку об’єкта моделювання протягом його життєвого циклу. Тобто, це правила у вигляді ліній. за допомогою яких можна побачити контролер переходу з одного стану до іншого.

Як вказано на схемі, що зображена нижче, після авторизації можливі такі дії:

– отримання унікального пароля;
– створення хешу, а потім унікального пароля;
– можливість зміни пароля;

Діаграма автоматів програмного додатку генерації та збереження паролів "Safe Password Storage" зображено на рис. 2.3.

Рисунок 2.3 – Діаграма станів автоматів програмного додатку генерації та збереження паролів "Safe Password Storage"

2.2 Модель аналізу предметної області програмного додатку генерації та збереження паролів "Safe Password Storage"

2.2.1 Діаграма класів аналізу програмного додатку генерації та збереження паролів "Safe Password Storage"

При моделюванні об’єктно-орієнтованих систем використовується діаграма класів аналізу. Діаграма класів являє собою сукупність елементів моделі, що відображають операції та відношення що їх з’єднують.

Діаграма класів аналізу програмного додатку генерації та збереження паролів "Safe Password Storage" зображено на рис. 2.4.

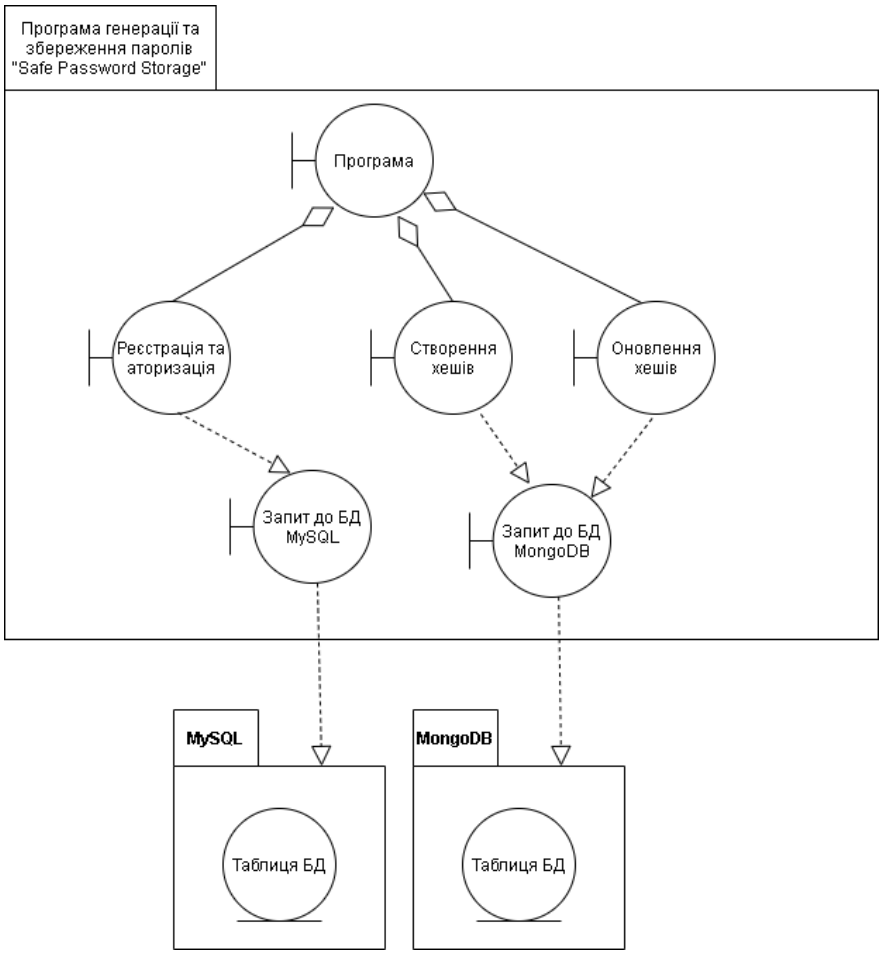


Рисунок 2.4 – Діаграма класів аналізу програмного додатку генерації та збереження паролів "Safe Password Storage"

2.2.2 Діаграми послідовності.

Діаграми послідовності відображають взаємодію об’єктів моделювання за часом. Об’єкт моделювання представлений у вигляді прямокутника, що розташований зверху пунктирної лінії – лінії життя моделі об’єкту.

Діаграми послідовності програмного додатку генерації та збереження паролів "Safe Password Storage" зображено на рис. 2.5 – 2.6.

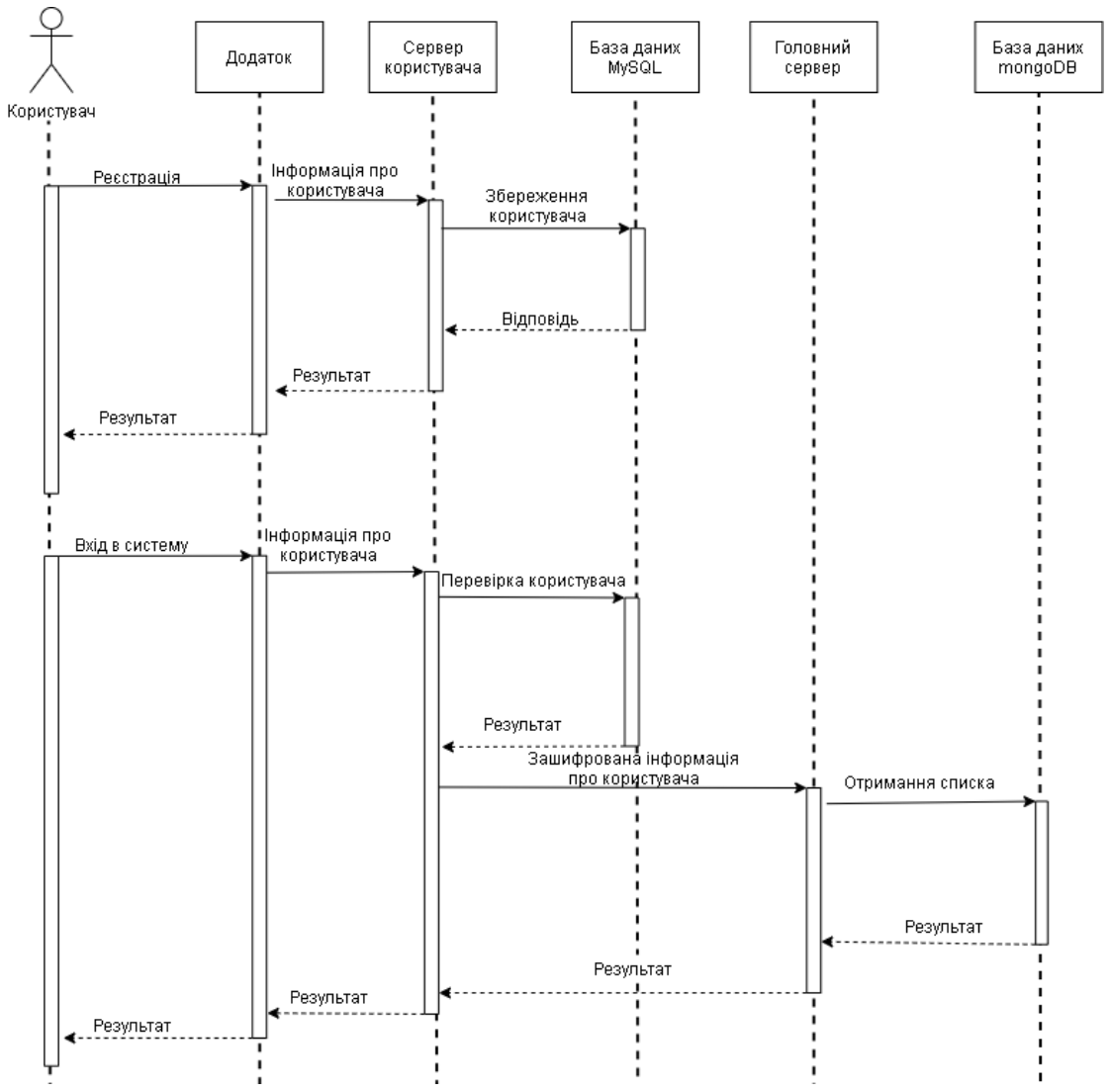


Рисунок 2.5 – Діаграма послідовності програмного додатку генерації та збереження паролів "Safe Password Storage"

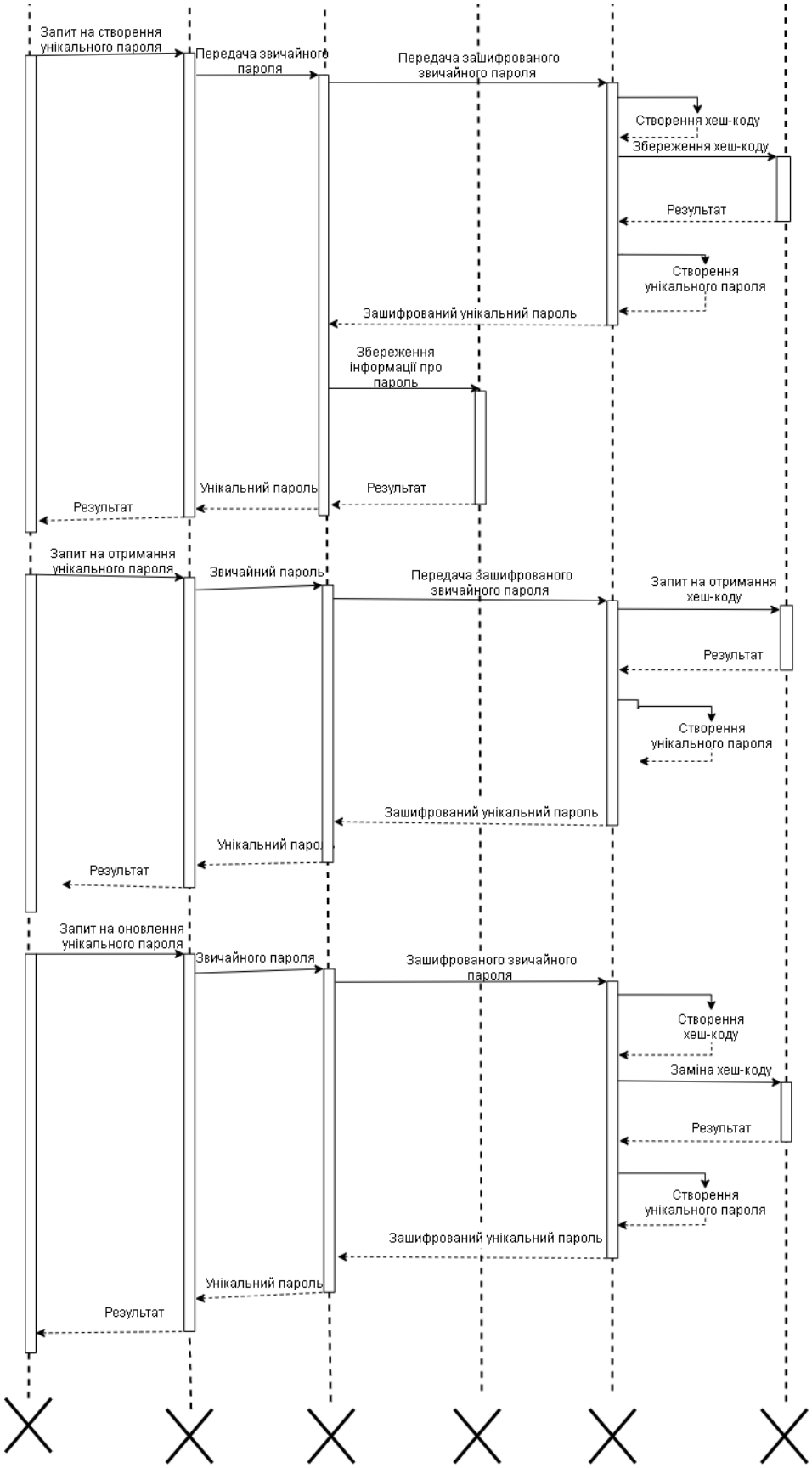


Рисунок 2.6 – Діаграма послідовності програмного додатку генерації та збереження паролів "Safe Password Storage"

2.2.3 Діаграми комунікації.

Діаграми комунікації містить інформацію про шлях за допомогою якого відбувається взаємодія між модулями програмного продукту.

Діаграму комунікації зображено на рис. 2.7

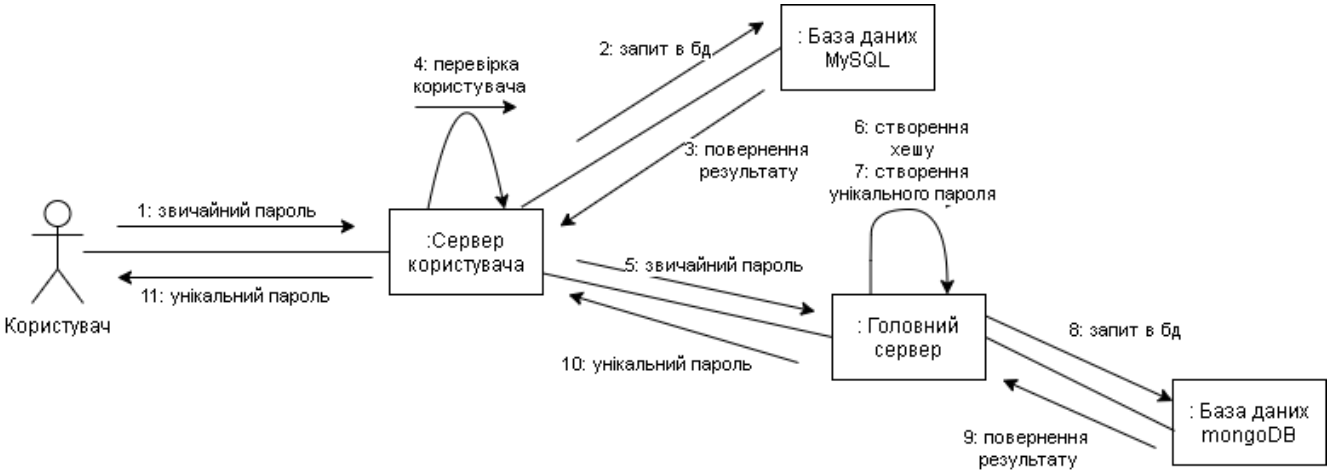


Рисунок 2.7 – Діаграма комунікації

2.3. Модель проектування програмного додатку генерації та збереження паролів "Safe Password Storage"

2.3.1 Діаграми класів.

Діаграми класів являє собою структуру моделі, що включає такі елементи як: класи та типи даних і відображає їх зміст. Вказана діаграма показує статичні структури моделі об'єктно-орієнтованого програмування [7].

У діаграмі класів використання визначені наступні основні класи сторінок:

– Авторизація. Клас відповідає за авторизацію користувача, має такі атрибути як логін та пароль. Після завершення операції “авторизація” перенаправляє користувача на головну сторінку.

– Головна сторінка. Клас відповідає за відображення списків web-сайтів, які створив користувач, а також має можливість перенаправити користувача на сторінку для створення паролю.
– Створення пароля. Клас відповідає за створення унікального пароля. Необхідними атрибутами для цього є назва web-сайту та звичайний пароль.
– Web-сайти. Клас відповідає за відображення інформації про сайт, а також має можливість перенаправити користувача на редагування пароля та на сторінку унікального пароля.
– Редагування пароля. Клас відповідає за редагування пароля. Необхідними атрибутами для цього є назва web-сайту та звичайний пароль.
– Унікальний пароль. Клас відповідає за отримання унікального пароля. Необхідним атрибутом для цього є звичайний пароль.

Діаграми класів програмного додатку генерації та збереження паролів "Safe Password Storage" зображено на рис. 2.8.

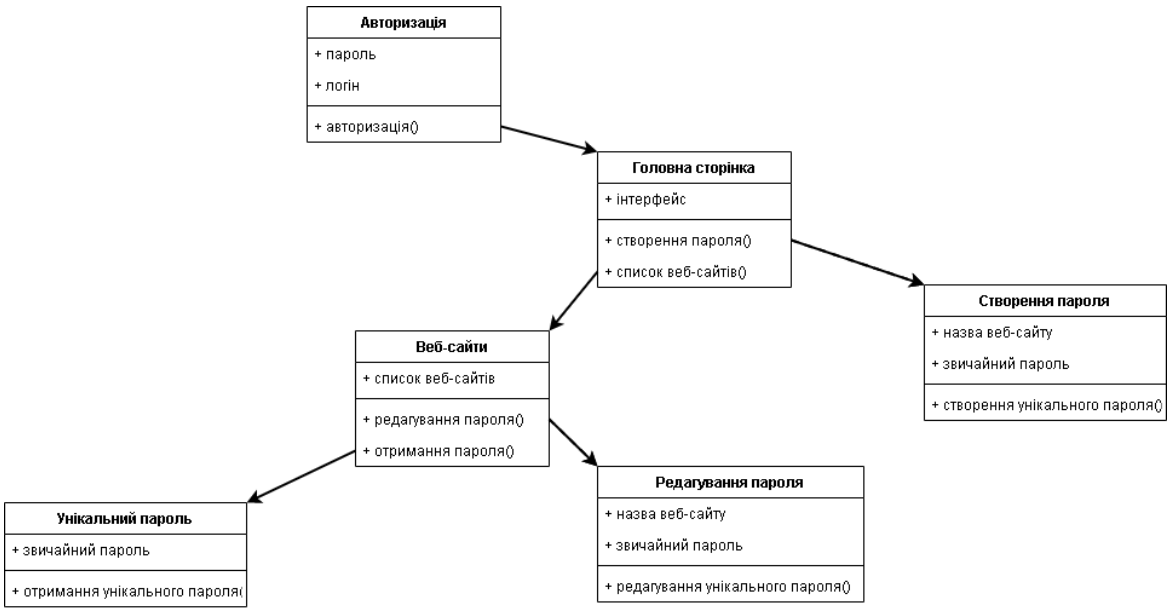


Рисунок 2.8 – Діаграма класів

2.3.2 Діаграми діяльності.

Діаграма діяльності – це діаграма, за допомогою якої звертаємо увагу на послідовність виконання якихось дій. Діаграма має зображення у вигляді графа, вершини якого визначають стани видів діяльності чи певних дій, а дуги – переходи від одного стану до іншого. За допомогою діаграми діяльності будь-хто може вибирати порядок дій [8].

Діаграми діяльності програмного додатку генерації та збереження паролів "Safe Password Storage" зображено на рис. 2.9.

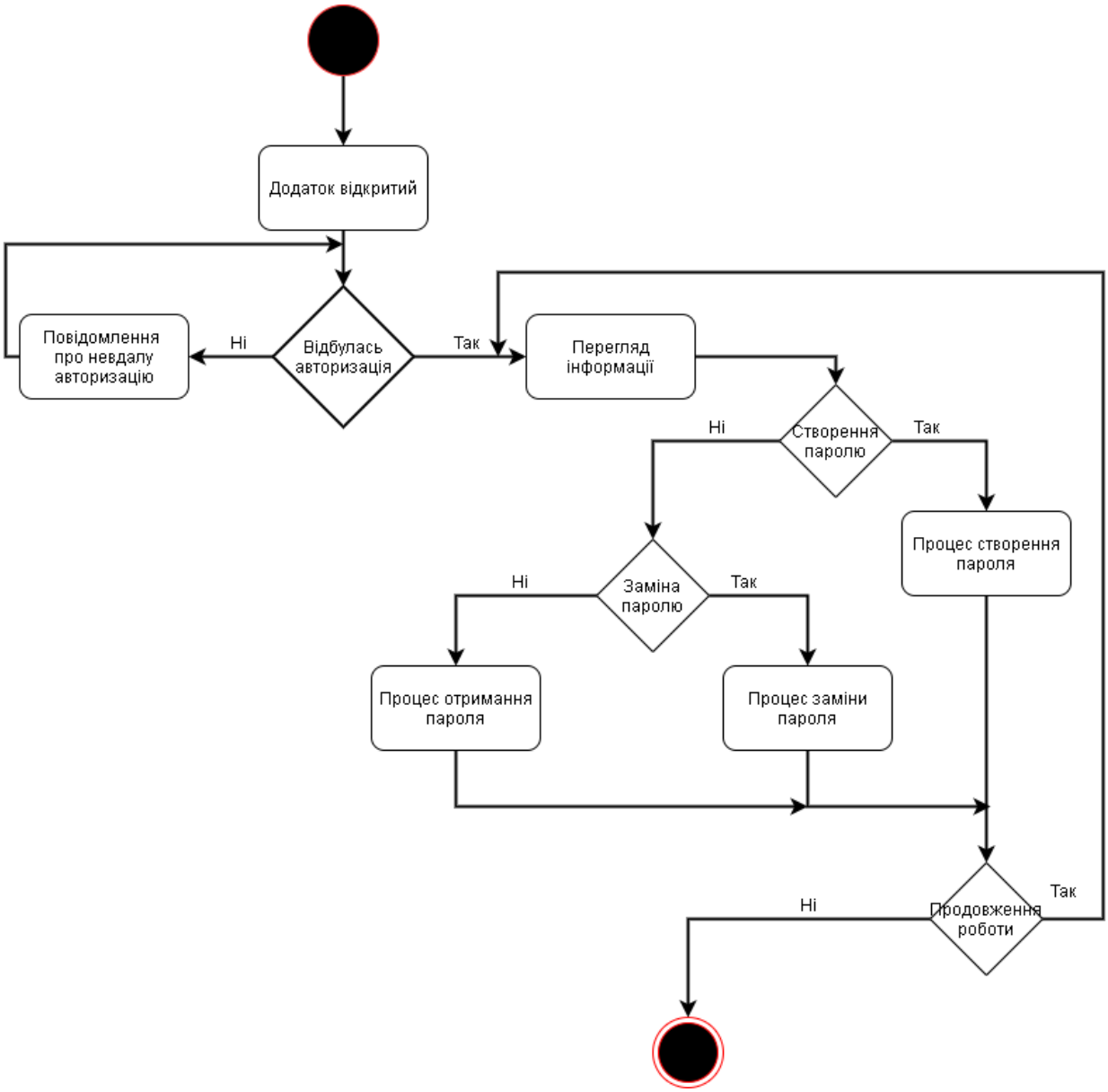


Рисунок 2.9 – Діаграма діяльності

2.4. Моделювання використання програмного додатку генерації та збереження паролів "Safe Password Storage"

2.4.1 Діаграма варіантів використання.

Для зображення в системі можливих відношень між акторами та прецедентами, будують діаграму варіантів використання.

Як показано на діаграмі, наведеній нижче, користувач має можливість на:

- авторизацію;
- реєстрацію;
- створення пароля;
- заміну пароля;
- перегляд паролів;
- отримання пароля;
- повідомлення про помилку на сервері.

В свою чергу адміністратор має можливість на:

- авторизацію;
- блокування користувача;
- розблокування користувача;
- підтримку роботи сервера.

Діаграма варіантів використання в UML представлена на рис. 2.10.

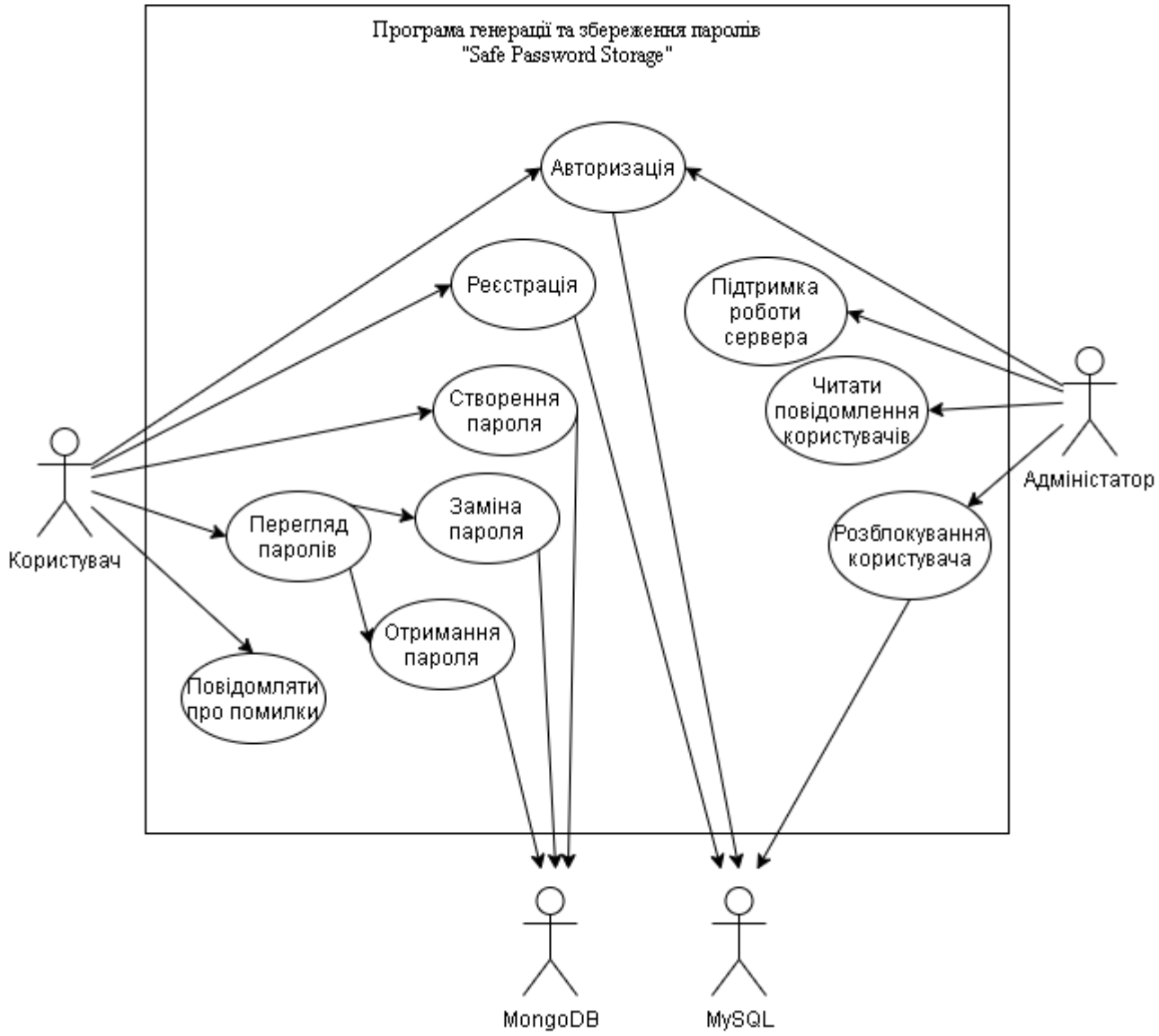


Рисунок 2.10 – Діаграма варіантів використання

2.5 Модель реалізації

2.5.1 Діаграма розгортання.

Діаграма розгортання застосовується для розроблення конфігурації програмної системи у вигляді графічного зображення процесів, пристроїв і зав’язків між ними. Ця діаграма завершує процес об’єктного програмування і є останнім етапом специфікації моделювання [9].

Діаграми розгортання програмного додатку генерації та збереження паролів "Safe Password Storage" зображено на рис. 2.11.

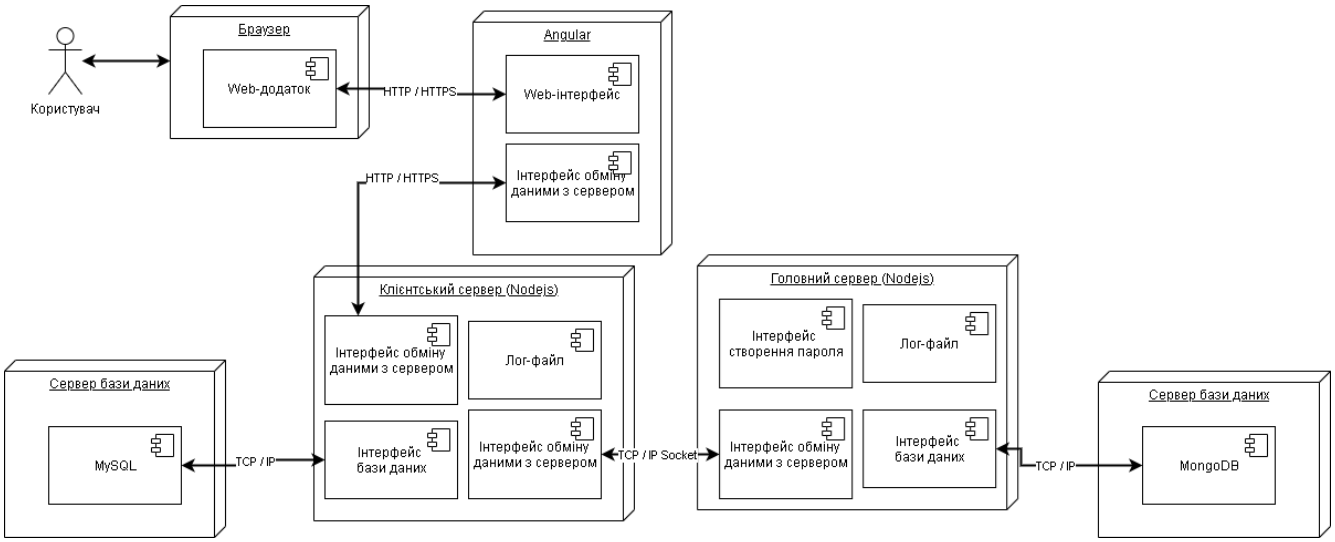


Рисунок 2.11 – Діаграма розгортання програмного додатку генерації та збереження паролів "Safe Password Storage"

2.6. Моделювання даних

2.6.1. Діаграма потоків даних

Діаграма потоків даних використовують для проведення аналізу інформаційних систем з метою зменшення їх об’єму, виявлення схожої інформації або її дублювання. Вона представлена у вигляді ієрархії діаграм, за допомогою яких можна спостерігати процес перетворення інформації з початку введення в систему до кінцевої видачі готової моделі користувачу [10].

Діаграми потоків даних програмного додатку генерації та збереження паролів "Safe Password Storage" зображено на рис. 2.12.

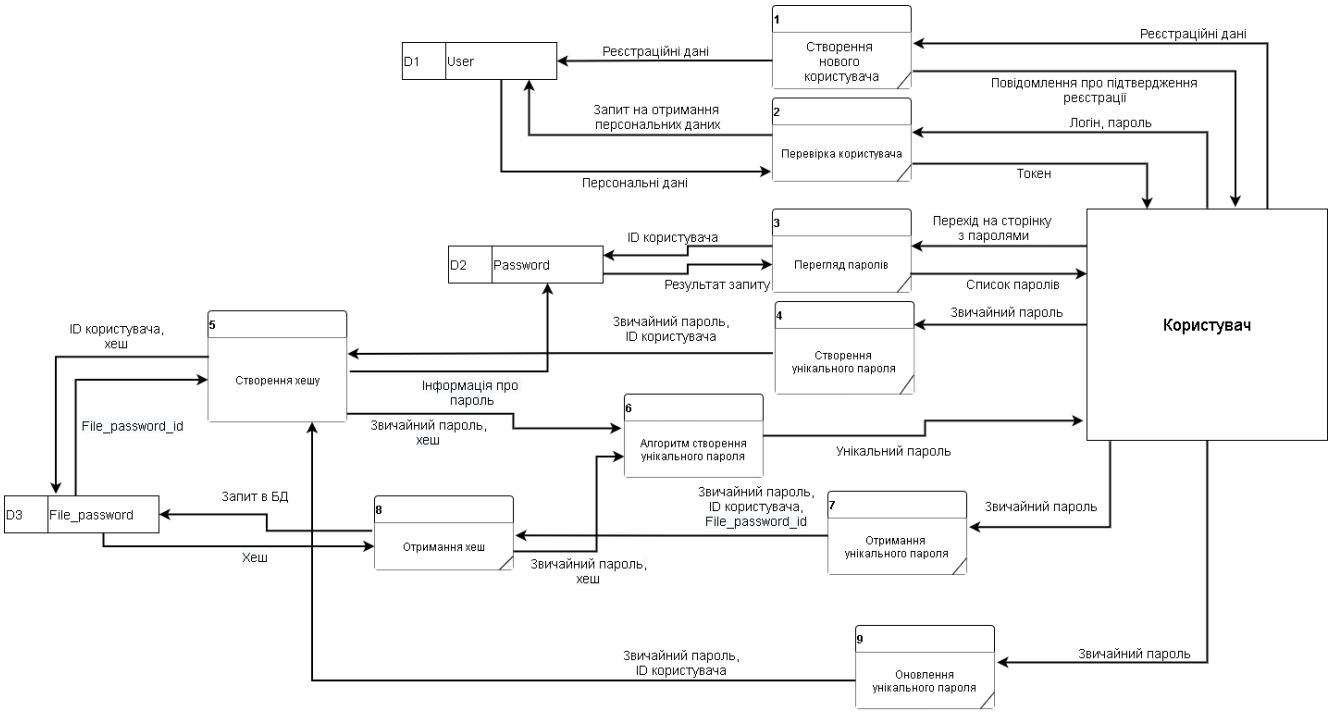


Рисунок 2.12 – Діаграма потоків даних

2.6.1. Логічна модель даних

Логічна модель даних – це основне джерело інформації на етапі фізичного проектування. З її допомогою можливо проаналізувати будь-які внесені до бази даних зміни й оцінити їх вплив на прикладі розробленої програми [11].

Під час роботи сервера на базі Node.js будуть використовувати дві бази даних: MySQL та MongoDB. База даних MySQL потрібна для зберігання інформації про користувача, а MongoDB - для зберігання файлу, який містить всю необхідну інформацію для створення унікального пароля.

Відомості про типи даних та ключові атрибути наведені нижче у вигляді таблиць (табл. 2.1 – 2.7).

Таблиця 2.1 – Список всіх сутностей бази даних

Users	Користувачі
Blockedusers	Заблоковані користувачі

Продовження таблиці 2.1

Passwords	Паролі
Messages	Повіломлення користувачів
Roles	Ролі користувачів
FilePassword	Файл для пароля

Таблиця 2.2 – Користувачі (Users)

№	Поле	Атрибут	Тип
1	id	Унікальний ідентифікатор користувача	UUID(36)
2	Email	Персональне ім'я користувача	VARCHAR(255)
3	Password	Пароль на вхід в систему	VARCHAR(255)
4	Username	Ім’я користувача	VARCHAR(255)

Таблиця 2.3 – Заблоковані користувачі (Blockedusers)

№	Поле	Атрибут	Тип
1	id	Унікальний ідентифікатор	UUID(36)
2	Description	Опис причини блокування	VARCHAR(250)
3	UserId	Унікальний ідентифікатор користувача	UUID(36)
4	UserName	Ім’я користувача	VARCHAR(255)

Таблиця 2.4 – Паролі (Passwords)

№	Поле	Атрибут	Тип
1	id	Унікальний ідентифікатор пароля	UUID(36)
2	Name	Ім’я пароля	VARCHAR(255)
3	File_password_Id	Унікальний ідентифікатор файлу пароля	VARCHAR(255)
4	Web_site_Id	Унікальний ідентифікатор сайту	VARCHAR(255)
5	User_Id	Унікальний ідентифікатор користувача	UUID(36)

Таблиця 2.5 – Повіломлення користувачів (Messages)

№	Поле	Атрибут	Тип
1	id	Унікальний ідентифікатор повідомлення	UUID(36)
2	Name	Ім’я повідомлення	VARCHAR(255)
3	UserId	Унікальний ідентифікатор користувача	UUID(36)
4	UserName	Ім’я користувача	VARCHAR(255)
5	Description	Опис причини повідомлення	VARCHAR(250)

Таблиця 2.6 – Ролі користувачів (Roles)

№	Поле	Атрибут	Тип
1	id	Унікальний ідентифікатор ролі	UUID(36)
2	Name	Ім’я ролі	VARCHAR(255)

Таблиця 2.7 – Файл для пароля (FilePassword)

№	Поле	Атрибут	Тип
1	_id	Унікальний ідентифікатор файлу пароля	OBJECTID
2	Id_user	Унікальний ідентифікатор користувача	STRING
3	webSites	Об’єкт, який зберігає всю необхідну інформацію для створення пароля	OBJECT
4	webSites._id	Унікальний ідентифікатор сайту	OBJECTID
5	webSites.hash	Унікальний хеш	STRING
6	webSites.test	Значення для тестування пароля на вірність	STRING
7	webSites.numbers	Масив значення для розміщення числових значень в паролі	[NUMBER]

Логічна модель даних програмного додатку генерації та збереження паролів "Safe Password Storage" зображено на рисунках 2.13 – 2.14.

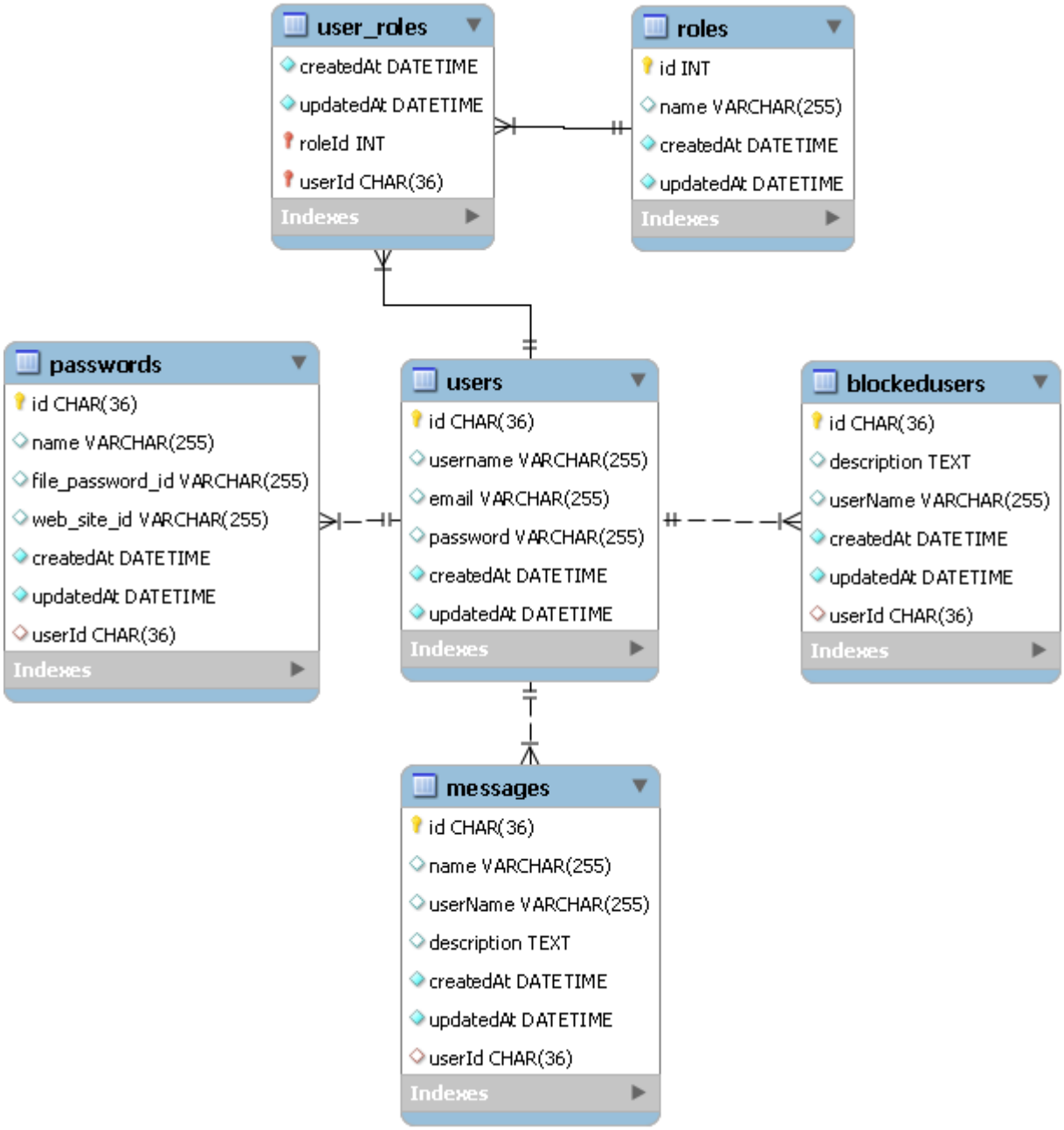


Рисунок 2.13 – Логічна модель даних MySQL

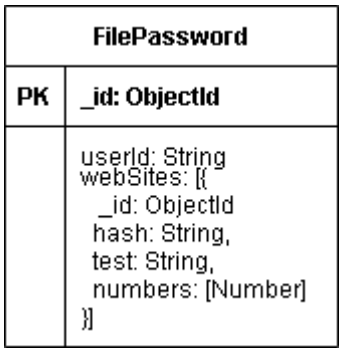


Рисунок 2.13 – Логічна модель даних MongoDB

3. РОЗРОБКА WEB-ДОДАТКУ

3.1 Створення проекту

Під час розробки програмного продукту генерації та збереження паролів "Safe Password Storage" використовувались програмний засіб Visual Studio Code (рис 3.1).

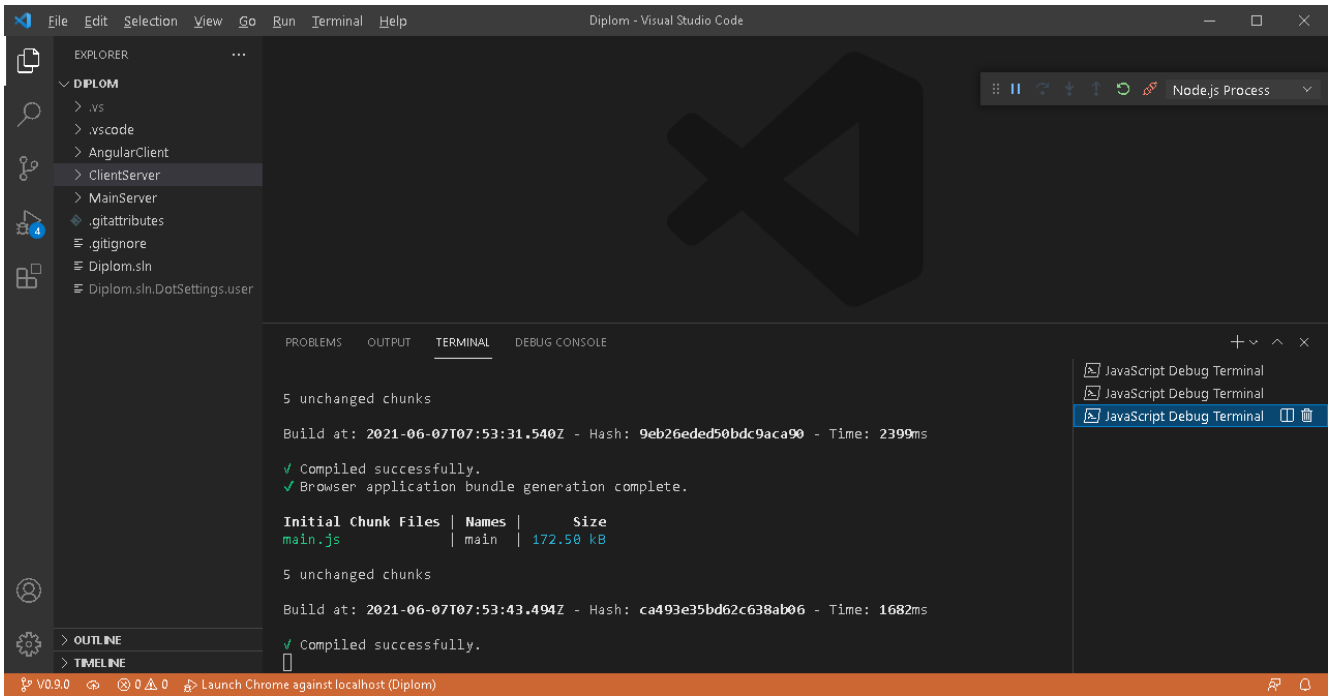


Рисунок 3.1 – Вікно програми Visual Studio Code

Для зручної налагодження програми були використані розширення в програмі Visual Studio Code (рис 3.2).

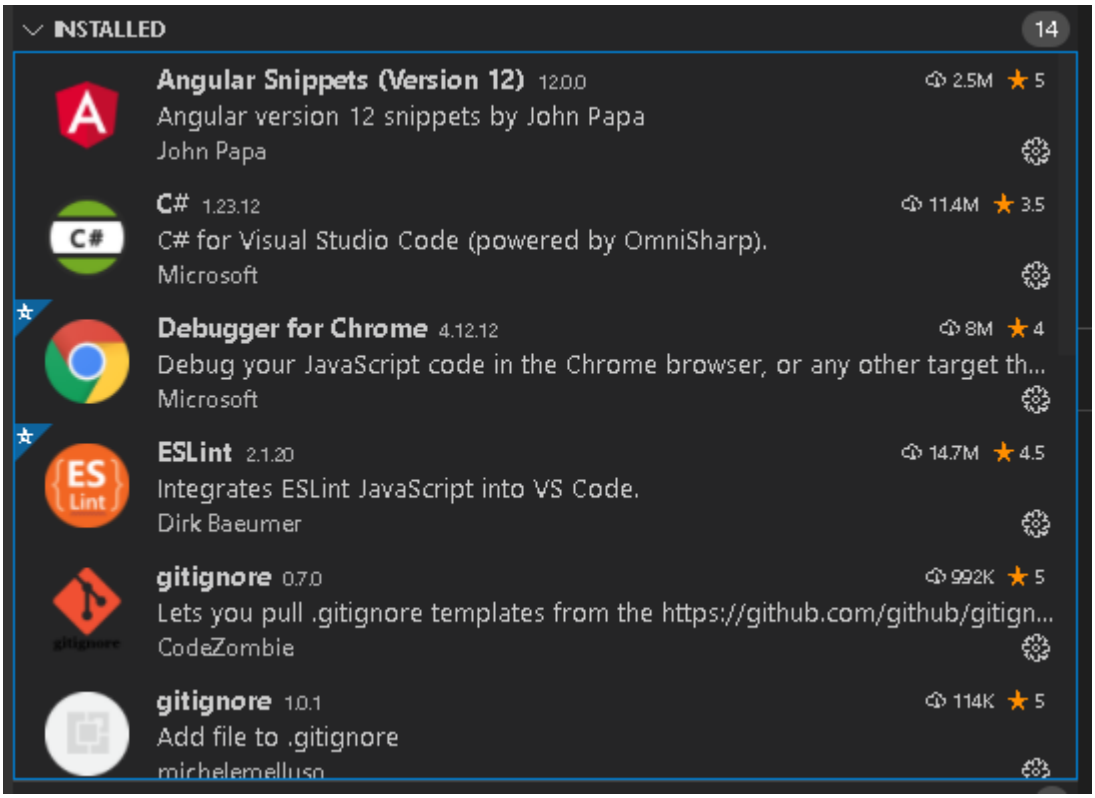


Рисунок 3.2 – Перелік встановлених розширень

Після цього в файлі проекту створено такі теки:

– AngularClient, що містить фреймворк Angular для взаємодії з клієнтом.

– ClientServer, який містить фреймворк Node.js, що відповідає за підключення до сервера MySQL, в якому зберігається інформація про користувача.

– MainServer також використовує фреймворк Node.js, що відповідає за підключення до сервера MongoDB, в якому зберігається інформація про хеші паролів.

3.1.1. Структура проекту AngularClient

Інформація про створені компоненти та сервіси наведені нижче у вигляді таблиці (табл. 3.1).

Таблиця 3.1 – Структура проекту AngularClient

№	Назва класу	Опис	Тип
1	AuthInterceptor	Відповідає за перетворення HTTP-запитів перед їх відправленням на сервер.	Interceptor
2	HttpErrorInterceptor	Відповідає за перевірку отриманого результату з сервера.	Interceptor
3	AuthService	Ця послуга надсилає запити на реєстрацію.	Service
4	TokenStorageService	Відповідає за управління інформацією про користувача. Інформацію зберігає в браузері.	Service
5	UserService	Ця послуга надає методи доступу до загальнодоступних та захищених ресурсів.	Service
6	BlockedComponent	Цей компонент відображає список заблокованих користувачів.	Component
7	BoardUserComponent	Цей компонент відображає список паролів, які створив користувач. В ньому користувач має змогу створити, отримати та відредагувати унікальний пароль.	Component
8	BugReportSystemComponent	Цей компонент відображає список повідомлень про помилки користувачів.	Component

Продовження таблиці 3.1

№	Назва класу	Опис	Тип
9	BugReportUsersComponent	Цей компонент відображає список повідомлень про помилки, що виникли на сервері.	Component
10	HomeComponent	Цей компонент відображає стартову сторінку проекту.	Component
11	LoginComponent	Цей компонент відображає форму для авторизації.	Component
12	ProfileComponent	Цей компонент відображає основну інформацію про користувача.	Component
13	RegisterComponent	Цей компонент відображає форму для реєстрації.	Component
14	ReportComponent	Цей компонент відображає форму для створення повідомлення адміністратора про помилки.	Component
15	AppRoutingModule	Відповідає за маршрутизацію для web-додатку.	Module
16	AppComponent	Цей компонент є кореневим компонентом фреймворку Angular.	Component
17	AppModule	Це кореневий модуль фреймворку Angular.	Module

Структура проекту AngularClient зображена на рисунку 3.3.

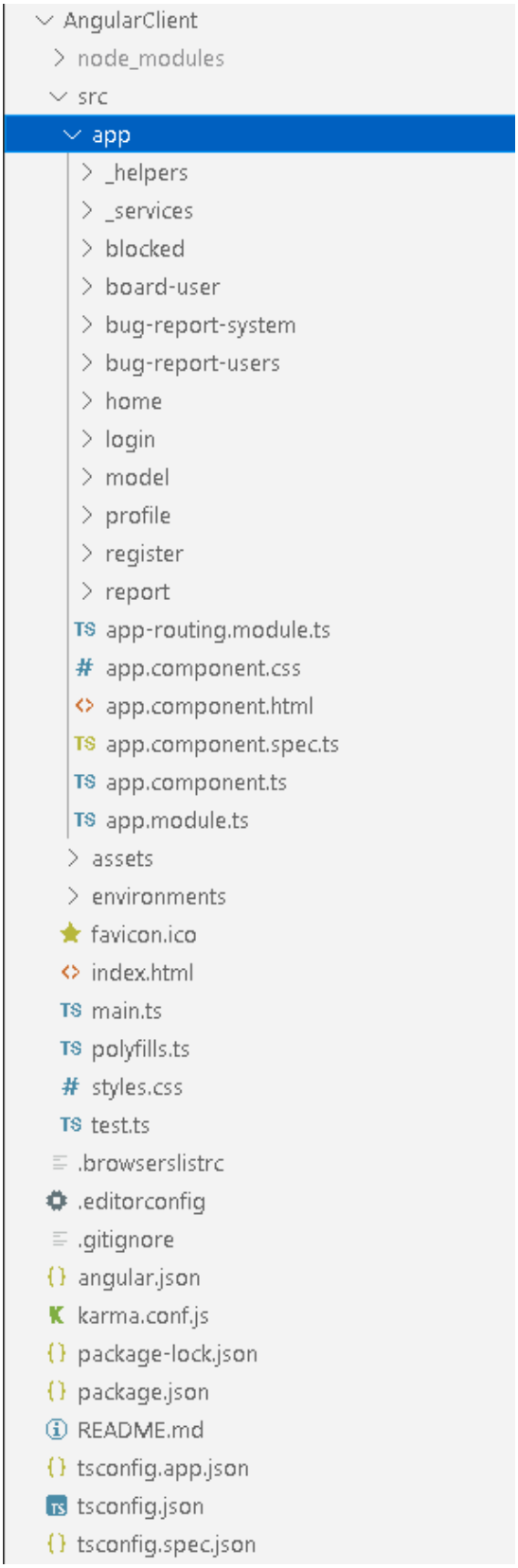


Рисунок 3.3 – Структура проекту AngularClient

3.1.2. Структура проекту ClientServer.

Структура каталогів для проекту ClientServer зображена на рисунку 3.4.

Рисунок 3.4 – Структура проекту ClientServer

Інформацію про створені файли проекту ClientServer наведено нижче у вигляді таблиці (табл. 3.2).

Таблиця 3.2 – Структура проекту ClientServer

№	Назва файлу	Опис
1	auth.config.js	Містить секретний ключ для побудови JWT аутентифікації.
2	Client.js	Шаблон класу для зберігання інформації користувача під час підключення через socket.
3	db.config.js	Містить параметри для підключення до бази даних MySQL та Sequelize.
4	auth.controller.js	Виконує обробку дії реєстрації та входу.
5	user.controller.js	Контролер для обробки запитів користувача.
6	rsa.js	Клас для створення закритих та відкритих ключів, шифрування та дешифрування інформації.
7	authJwt.js	Виконує перевірку користувача по ролі у базі даних.
8	socket.js	Містить функції для відправки запитів на головний сервер за допомогою бібліотеки socket.io-client.js.
9	verifySignUp.js	Виконує перевірки на повторюваність імені чи електронної пошти у базі даних.
10	models/index.js	За допомогою бібліотеки sequelize.js робить підключення до бази даних MySQL, використовуючи моделі файлів, які розташовані в такі models та створює таблиці з взаємозв’язками.
11	auth.routes.js	Містить запроси для реєстрації та авторизації.
12	user.routes.js	Містить основні запроси користувача та адміністратора.

Продовження таблиці 3.2

№	Назва файла	Опис
13	babel.config.js	Містить налаштування для використання бібліотеки babel фреймворком Node.js.
14	server.js	Виконує підключення всіх необхідних модулів та маршрутів.

3.1.3. Структура проекту MainServer

Структура каталогів для проекту MainServer зображена на рисунку 3.5.

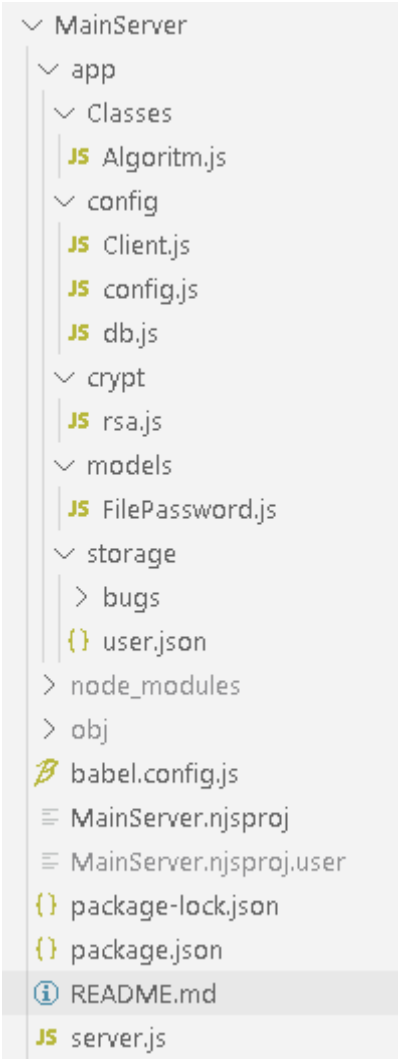


Рисунок 3.4 – Структура проекту MainServer

Інформація про створені файли проекту MainServer наведені нижче у вигляді таблиці (табл. 3.3).

Таблиця 3.3 – Структура проекту MainServer

№	Назва файлу	Опис
1	Algoritm.js	Містить метод створення унікального пароля.
2	Client.js	Шаблон класу для зберігання інформації користувача під час підключення через socket.
3	config.js	Містить параметри для підключення до бази даних MongoDB та mongoose.
4	db.js	Містить необхідні методи для взаємодії з базою даних MongoDB.
5	rsa.js	Клас для створення закритих та відкритих ключів, шифрування та дешифрування інформації.
6	FilePassword.js	Містить схему таблиці FilePassword.
7	user.json	Зберігає інформацію про кількість невірно введених паролів користувача.
8	babel.config.js	Містить налаштування для використання бібліотеки babel фреймворком Node.js.
9	server.js	Виконує підключення всіх необхідних модулів та маршрутів.

3.1.4. Реалізація авторизації та реєстрації на сервері

Для проведення аутентифікації в проекті використовується бібліотека jsonwebtoken.js, яка потрібна для верифікації тверджень. Авторизований користувач має токен, який зберігається в sessionStorage та перевіряється при кожному запиті. Коли час дії токена вийшов, клієнту потрібно знову авторизуватися для продовження роботи в системі.

Для формування ключа та безпечного зберігання паролів користувача в базі даних було використано бібліотеку bcryptjs.js. Для того щоб дізнатися пароль злоумисник використовує райдужні таблиці, тобто бази даних, в яких за пароль відповаідає обчислений хеш. Знайшли хеш – знайшли пароль. Для захисту від цих атак bcrypt використовує необоротні хеш-функції, які включають salt. Дана функція збільшує довжину та складність пароля, що робить попередні обчислення просто безглуздими а отже атаку неможливою.

3.1.5. Реалізація підключення клієнтського та головного сервера

Підключення та передача даних виконується за допомогою бібліотеки Socket.IO. Socket.IO – це JavaScript-бібліотека для web-застосунків і обміну даними в реальному часі. Складається з двох частин: клієнтської і серверної.

Було створено клас RSA, призначений для шифрування та дешифрування даних. За допомогою нього відбувається створення відкритого і закритого ключа. Послідовність підключення та передача даних між серверами наведено на рисунках 3.5-3.6.

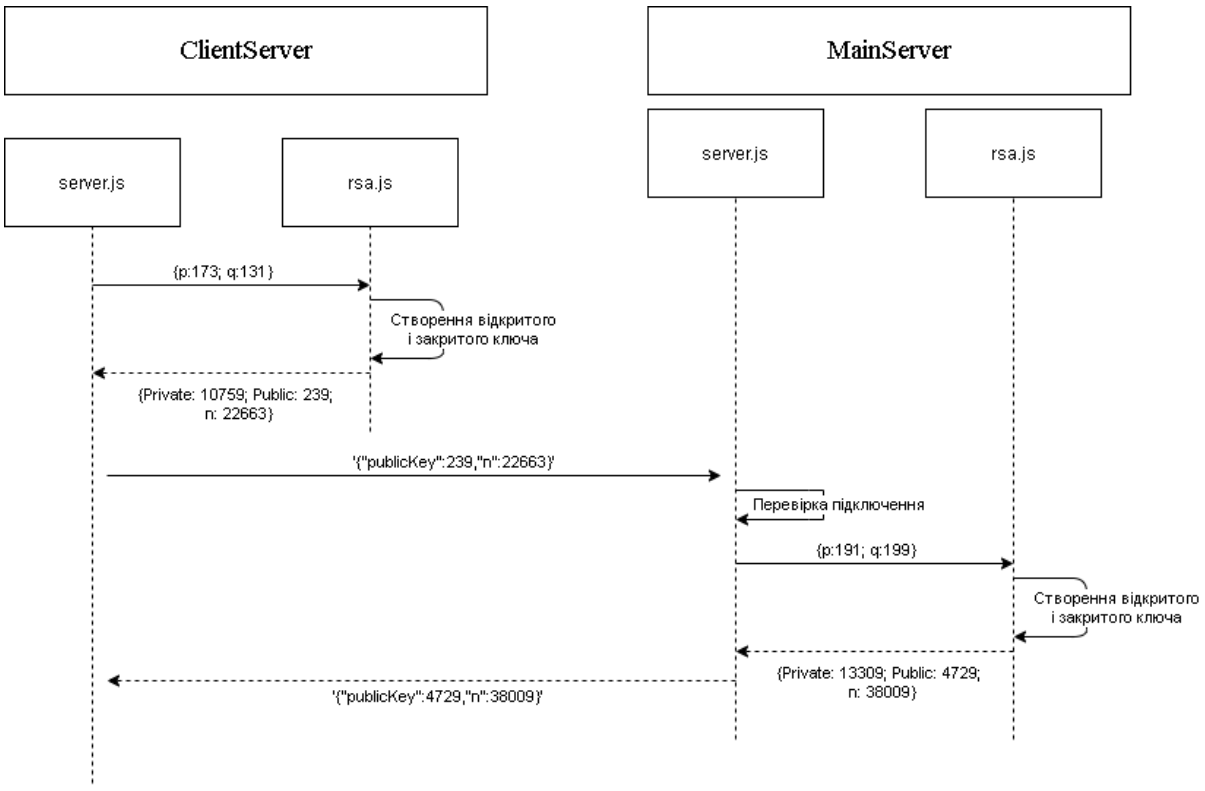


Рисунок 3.5 – Встановлення зв'язку між серверами

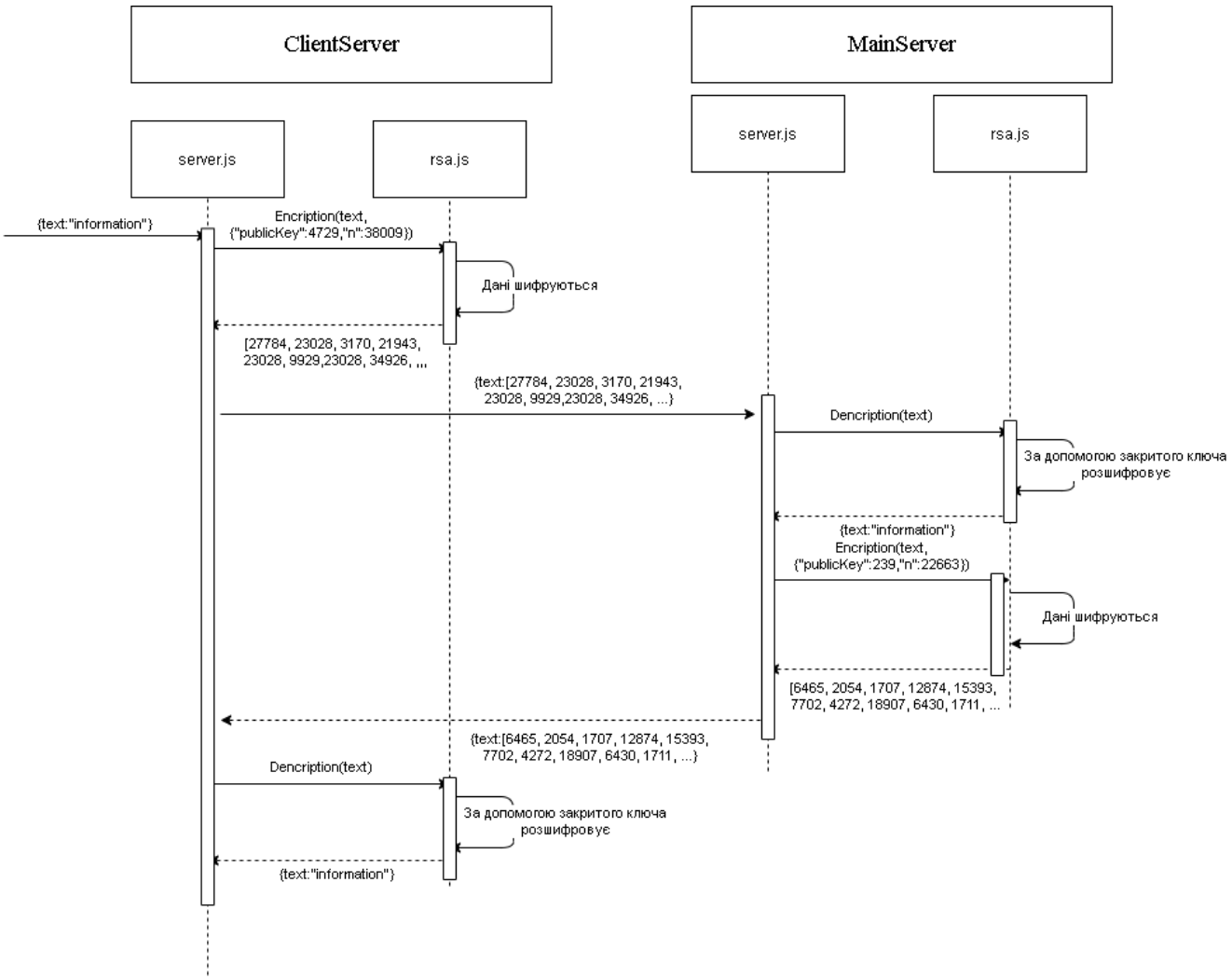


Рисунок 3.6 – Передача даних між серверами

3.1.5. Алгоритм роботи шифрування пароля

Алгоритм шифрування паролів був розроблений автором проекту, детальний опис роботи якого наведений нижче.

На початку роботи формується хеш та масив чисел для врахування позиції додаткової інформації в паролі. Далі виконується сумування кожного символу як код символу. Отриманий результат перемножується на кожний символ. Результат розбивається на символи та множиться з хешем. Після цього відбувається ділення результату на частини та проходить перевірка на кількість елементів в масиві.

Якщо умова перевірки невірна то зменшується або додається додаткова інформація до тих пір, поки умова не буде дійсною. Отриманий результат передається у функцію CreatePassword. В цій функції відбувається створення структури майбутнього пароля та за допомогою неї будуємо унікальний пароль.

Унікальність даного алгоритму формування пароля полягає в тому, що немає ніякої можливості відновити звичайний пароль. Блок-схеми кожного етапу формування пароля відображено на рисунках 3.7-3.12.

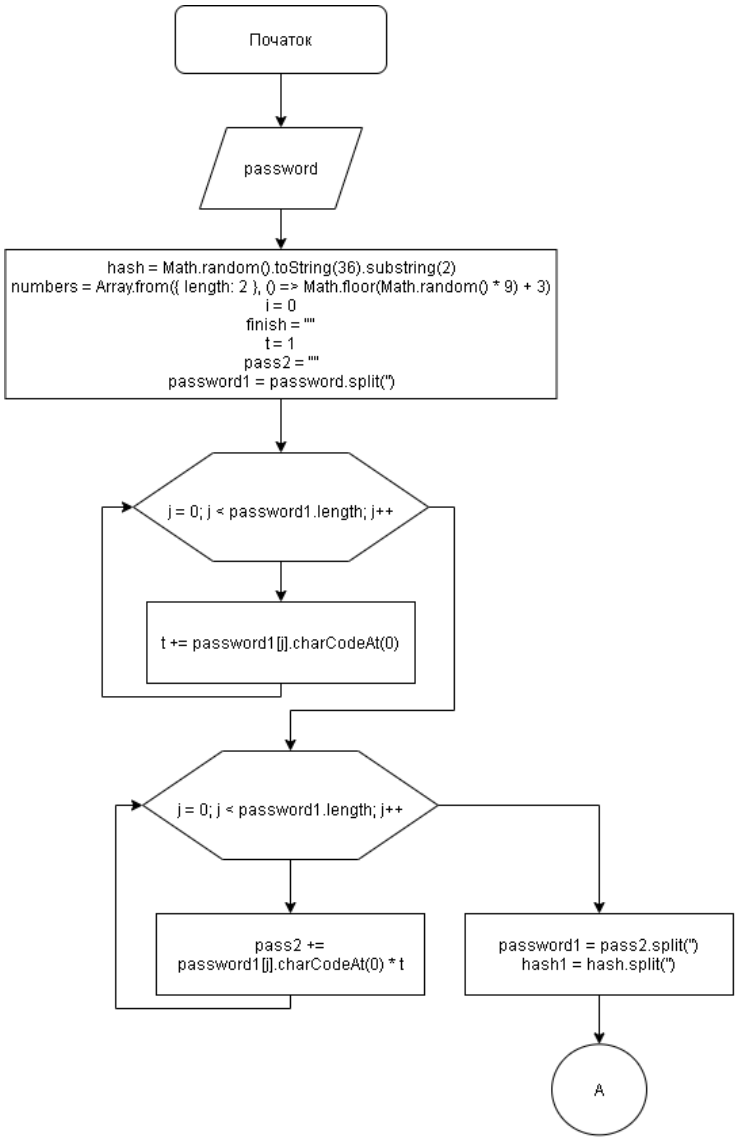


Рисунок 3.7 – Блок-схема стартового алгоритму

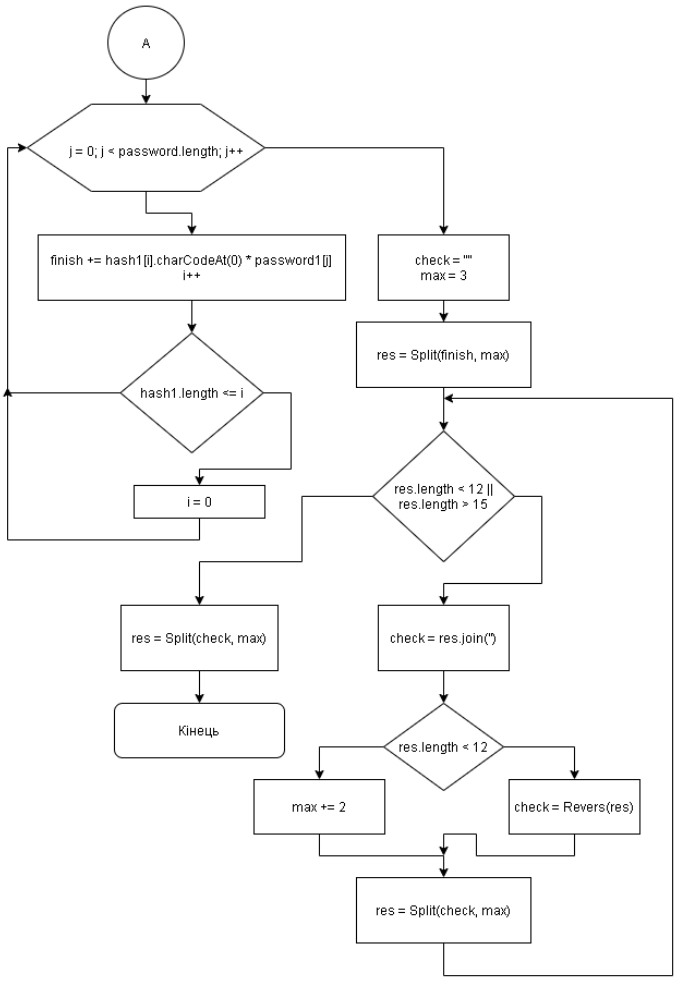


Рисунок 3.8 – Продовження блок-схеми стартового алгоритму

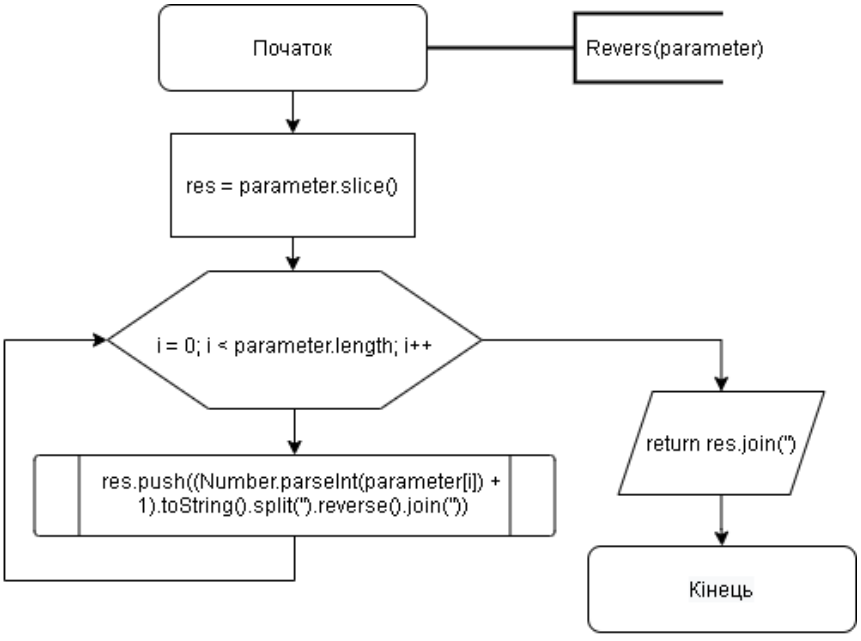


Рисунок 3.9 – Блок-схема функції Revers

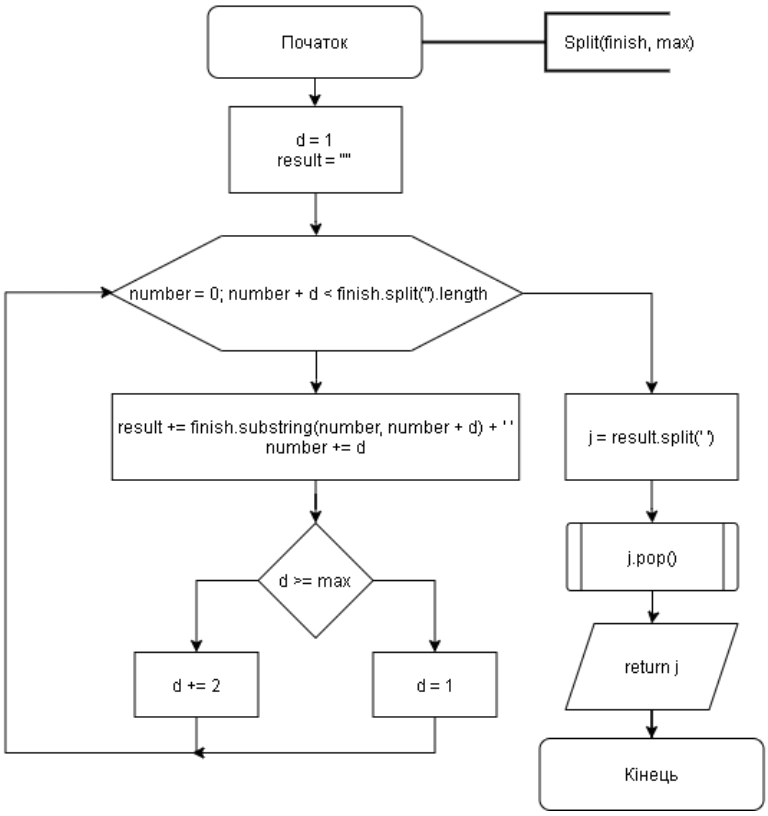


Рисунок 3.9 – Блок-схема функції Split

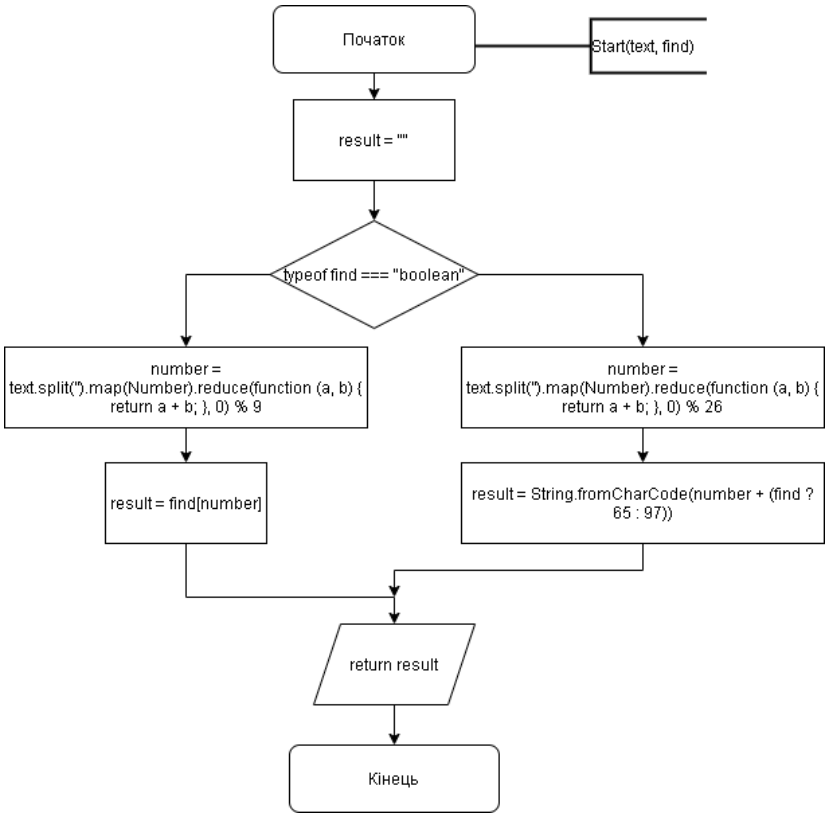


Рисунок 3.10 – Блок-схема функції Start

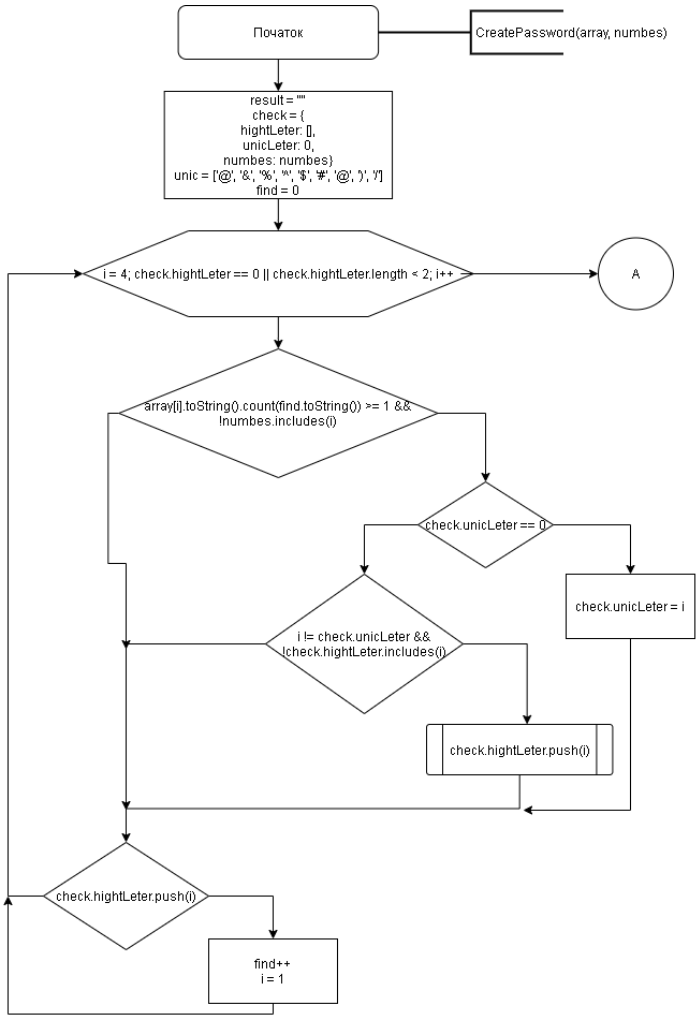


Рисунок 3.11 – Блок-схема функції CreatePassword

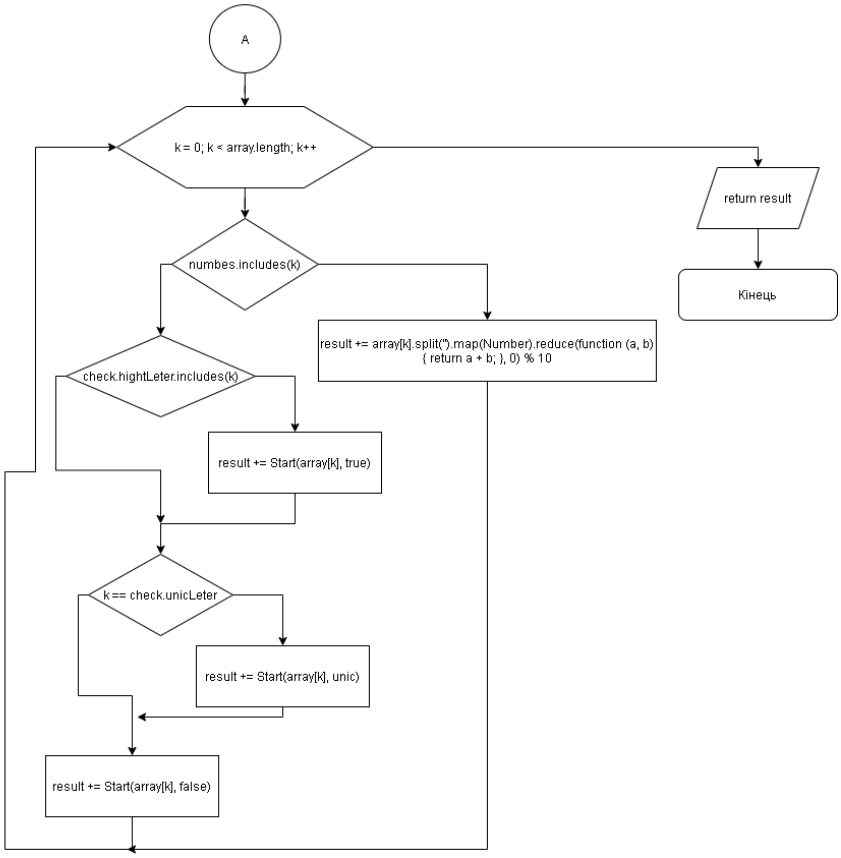


Рисунок 3.12 – Продовження блок-схеми функції CreatePassword

3.2 Результат виконаної роботи

У результаті виконання кваліфікаційної роботи бакалавра було досягнуто мету проекту–створення програми генерації та збереження паролів “Safe Password Storage”. Створено основний функціонал:

- головна сторінка сайту;
- механізм авторизації та реєстрації користувачів;
- можливість створення та редагування пароля;
- можливість створення повідомлення про помилки;
- адміністративна частина.

Головна сторінка сайту зображено на рисунку 3.13.

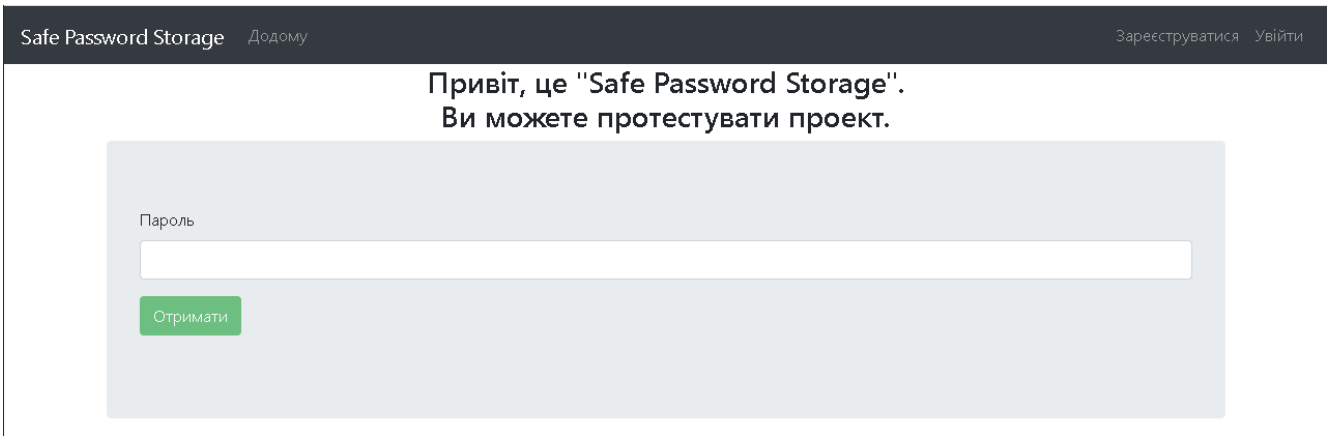


Рисунок 3.13 – Скріншот головної сторінки

Дане програмне забезпечення дає можливість неавторизованому користувачеві зробити тестову спробу перевірки алгоритму шифрування паролів шляхом введення будь якого пароля та в результаті отримати зразок нового унікального пароля (рис. 3.14).

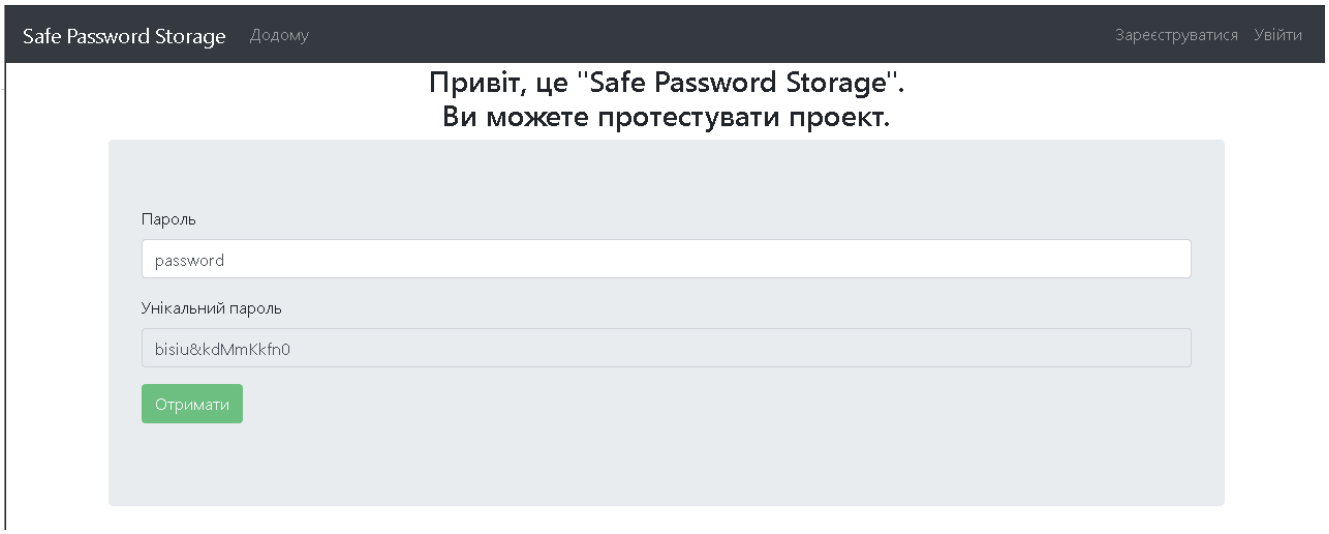


Рисунок 3.14 – Скриншот роботи алгоритму

Першим етапом роботи користувача в даному програмному забезпеченні є сторінки реєстрації та авторизації (рис. 3.15 – 3.16).

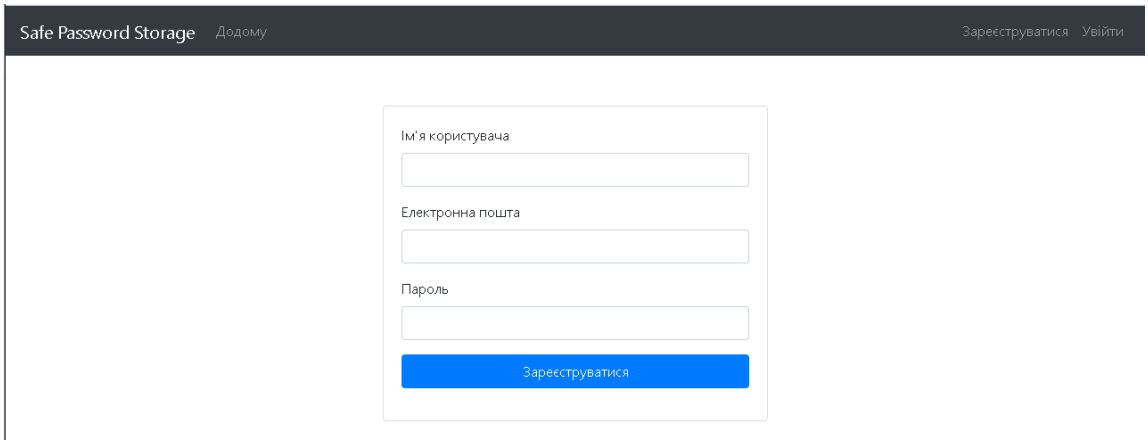


Рисунок 3.15 – Сторінка реєстрації

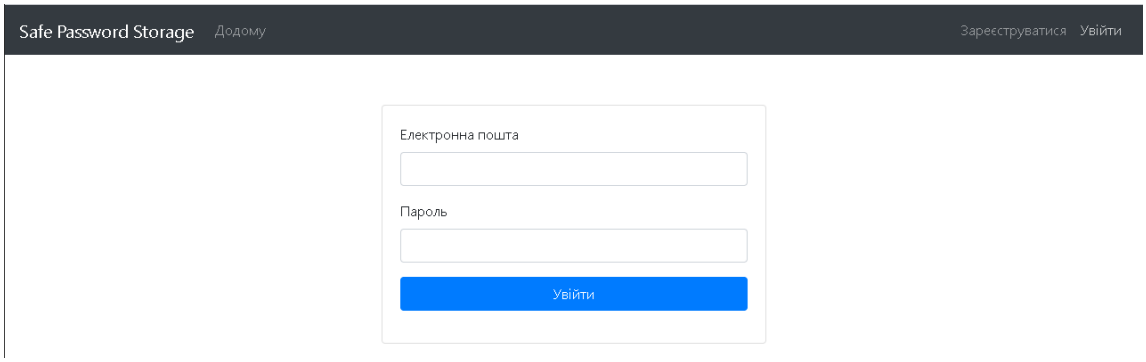


Рисунок 3.16 – Сторінка авторизації

Одним з функціоналів роботи сайту є перевірка на валідність введених даних клієнта (рис. 3.17).

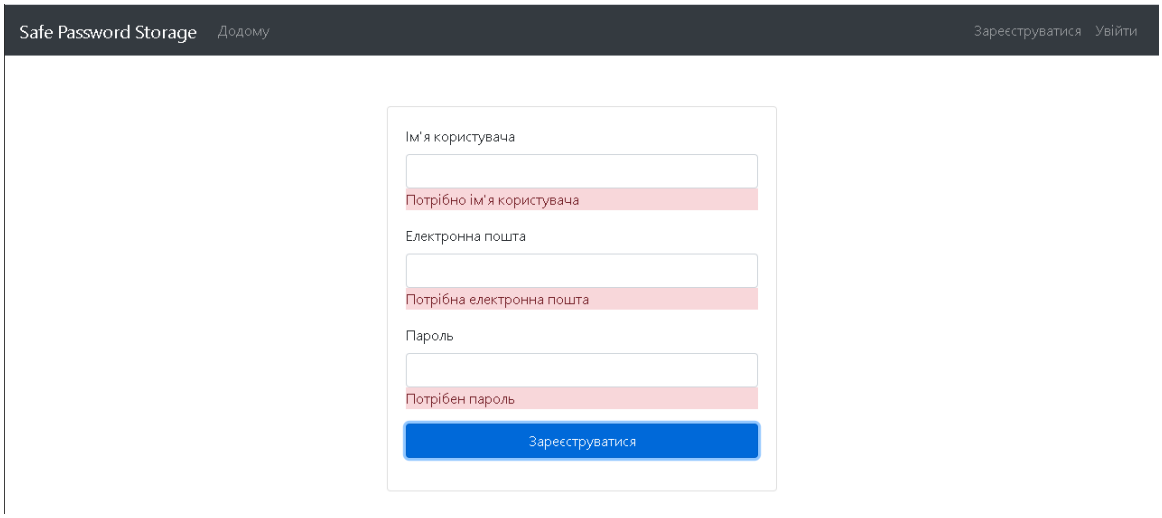


Рисунок 3.17 – Скріншот роботи перевірки на валідність

Ім’я та логін клієнта під час реєстрації проходять перевірку на сумісність в базі даних для виявлення подібності (рис. 3.18).

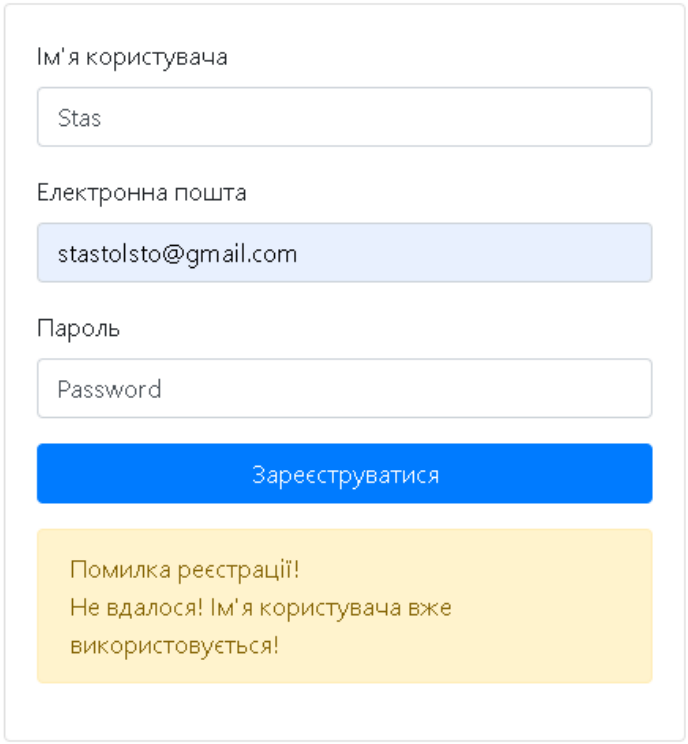


Рисунок 3.18 – Скріншот повідомлення про те, що ім’я вже існує

Після реєстрації користувач може здійснити вхід через сторінку авторизації (рис. 3.19 – 3.20).

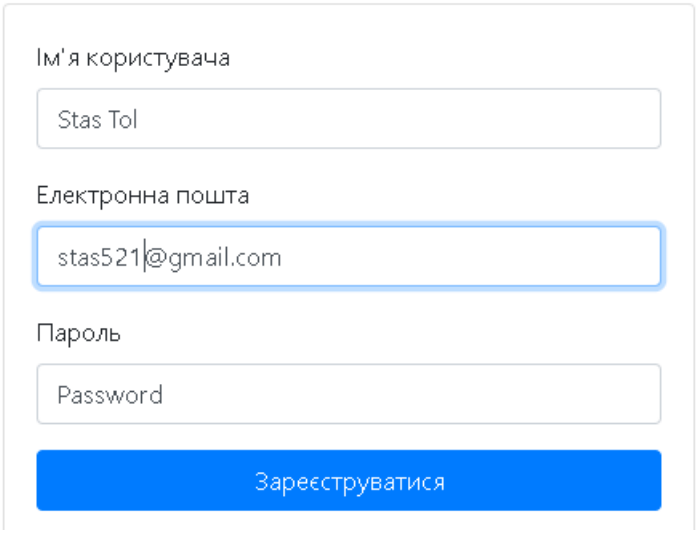


Рисунок 3.19 – Скріншот реєстрації користувача

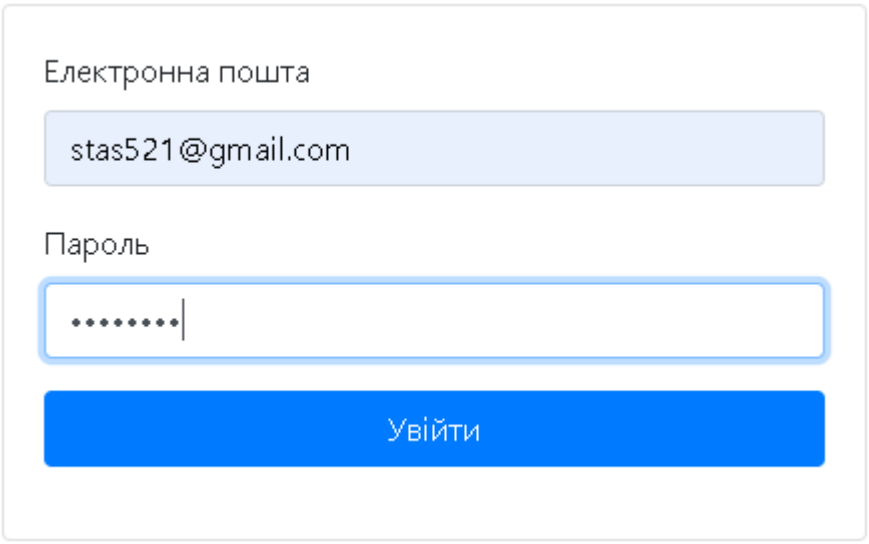


Рисунок 3.20 – Скріншот авторизації користувача

Після успішної авторизації користувач матиме змогу перейти на такі сторінки як створення паролю та звіту про помилку (рис. 3.21 – 3.22).

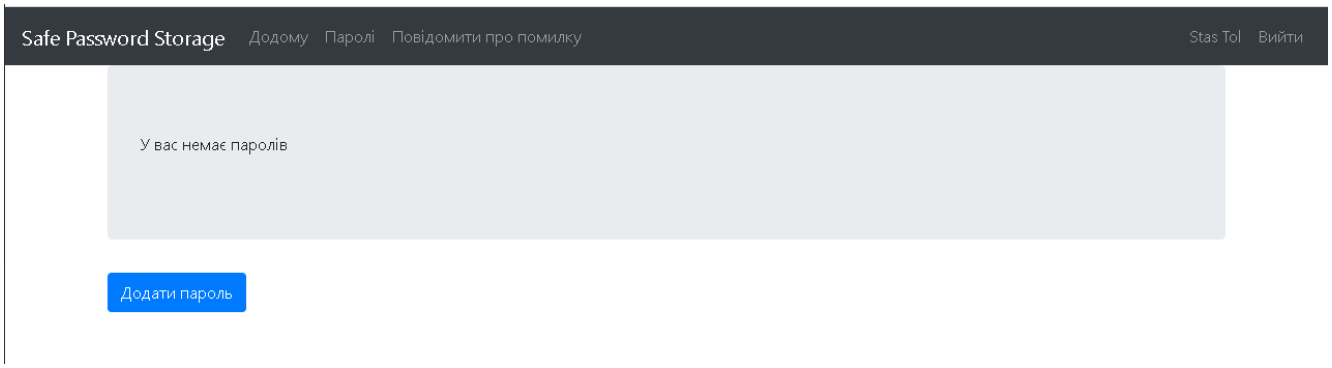


Рисунок 3.21 – Сторінка взаємодії з паролем

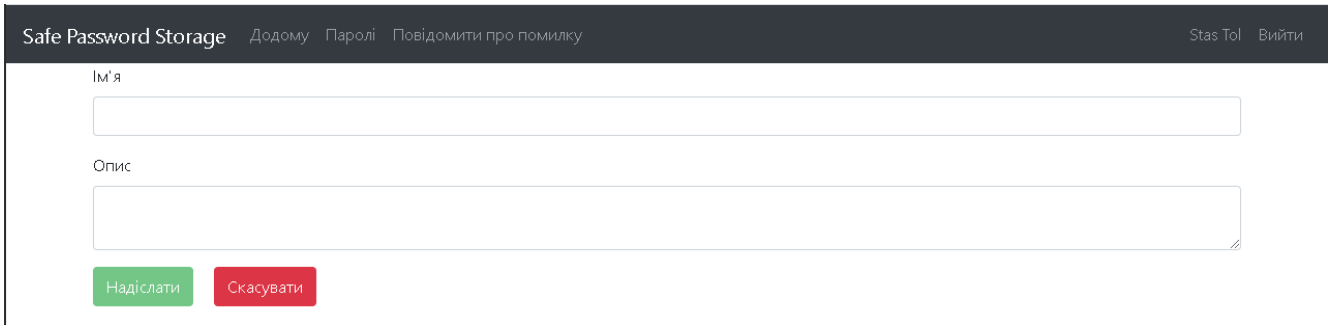


Рисунок 3.22 – Сторінка створення повідомлення про помилку

Користувач, який був більше ніж 15 хвилин на сайті, отримає повідомлення про те, що час роботи токену завершився (рис. 3.23).

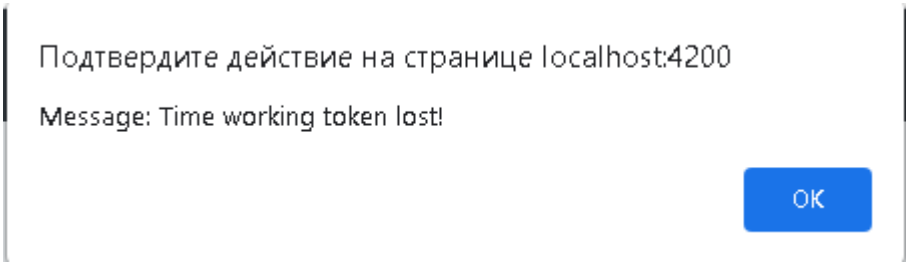


Рисунок 3.23 – Повідомлення про завершення роботи токена

На сторінці взаємодії з паролем користувачу спочатку потрібно створити пароль (рис. 3.24). Після підтвердження введених даних користувач отримає унікальний пароль (рис. 3.25).

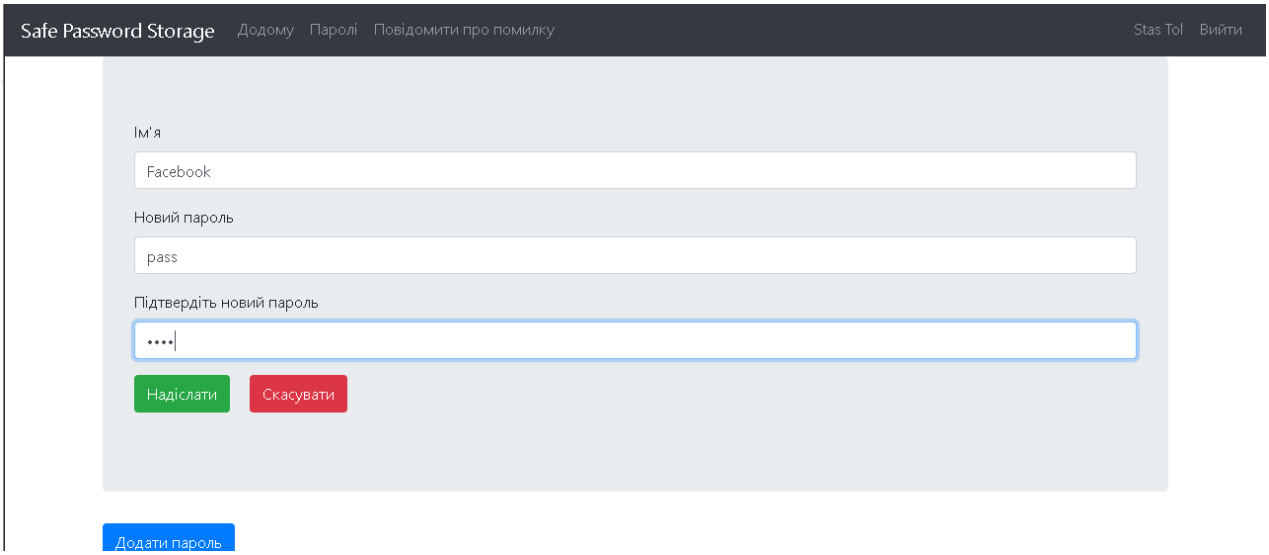


Рисунок 3.24 – Форма створення пароля

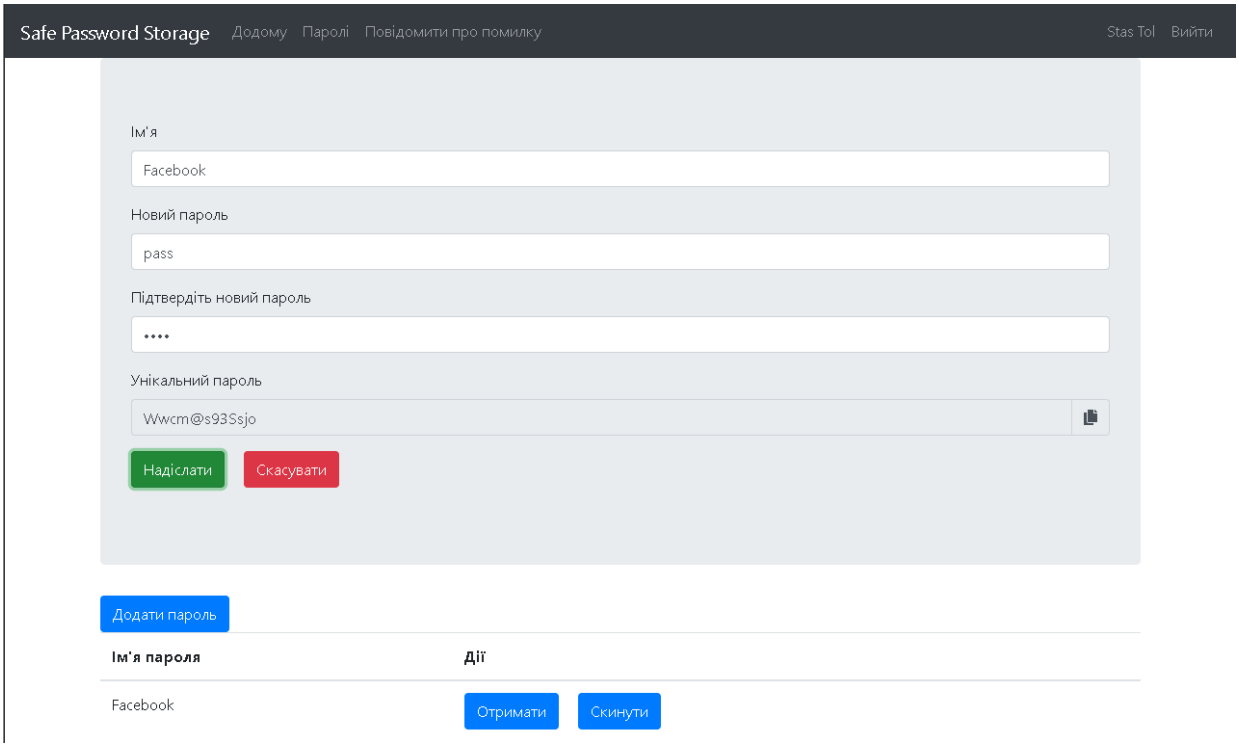


Рисунок 3.25 – Результат створення пароля

Після повторного завантаження сторінки з створеними паролями (рис. 3.26) користувач матиме змогу знову отримати унікальний пароль якщо введе вірно дані звичайного пароля (рис. 3.27).

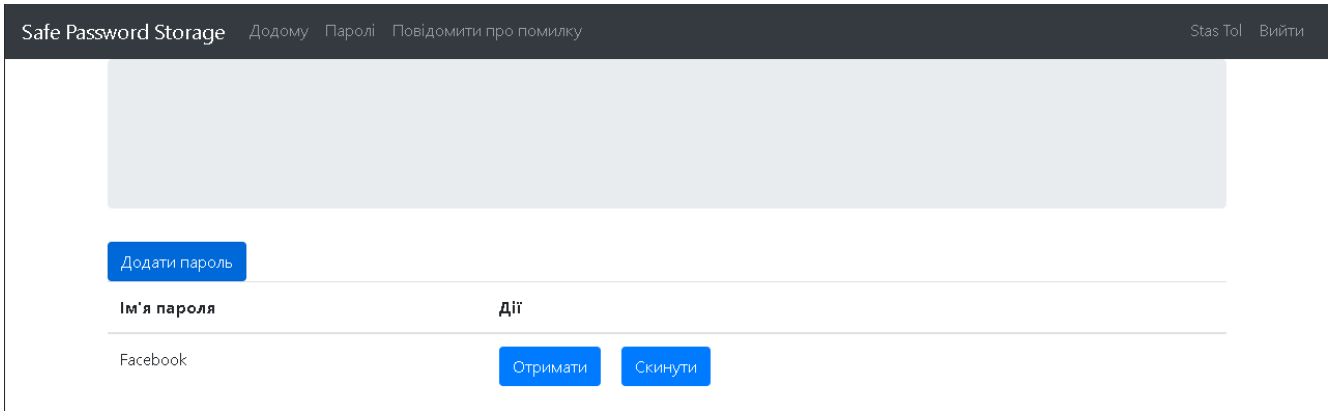


Рисунок 3.26 – Оновлена сторінка паролів

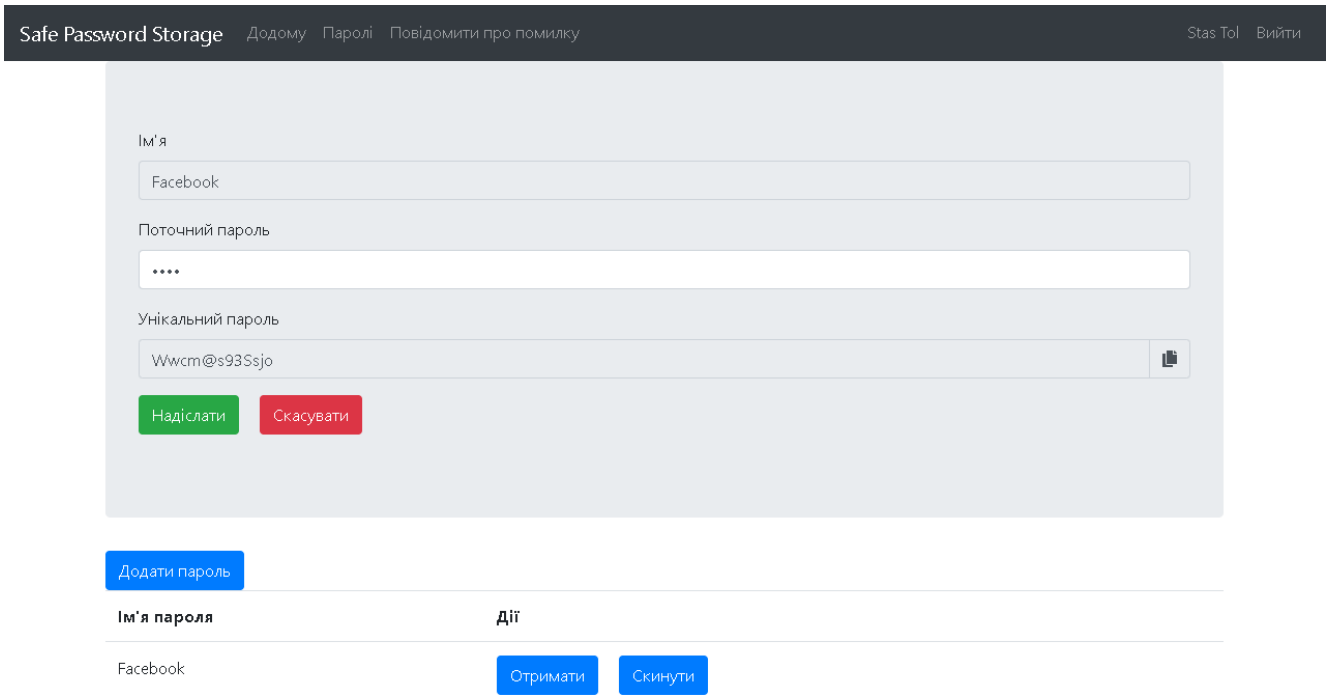


Рисунок 3.27 – Результат отримання пароля

Користувач має змогу змінити існуючий пароль (рис. 3.28 – 3.29).

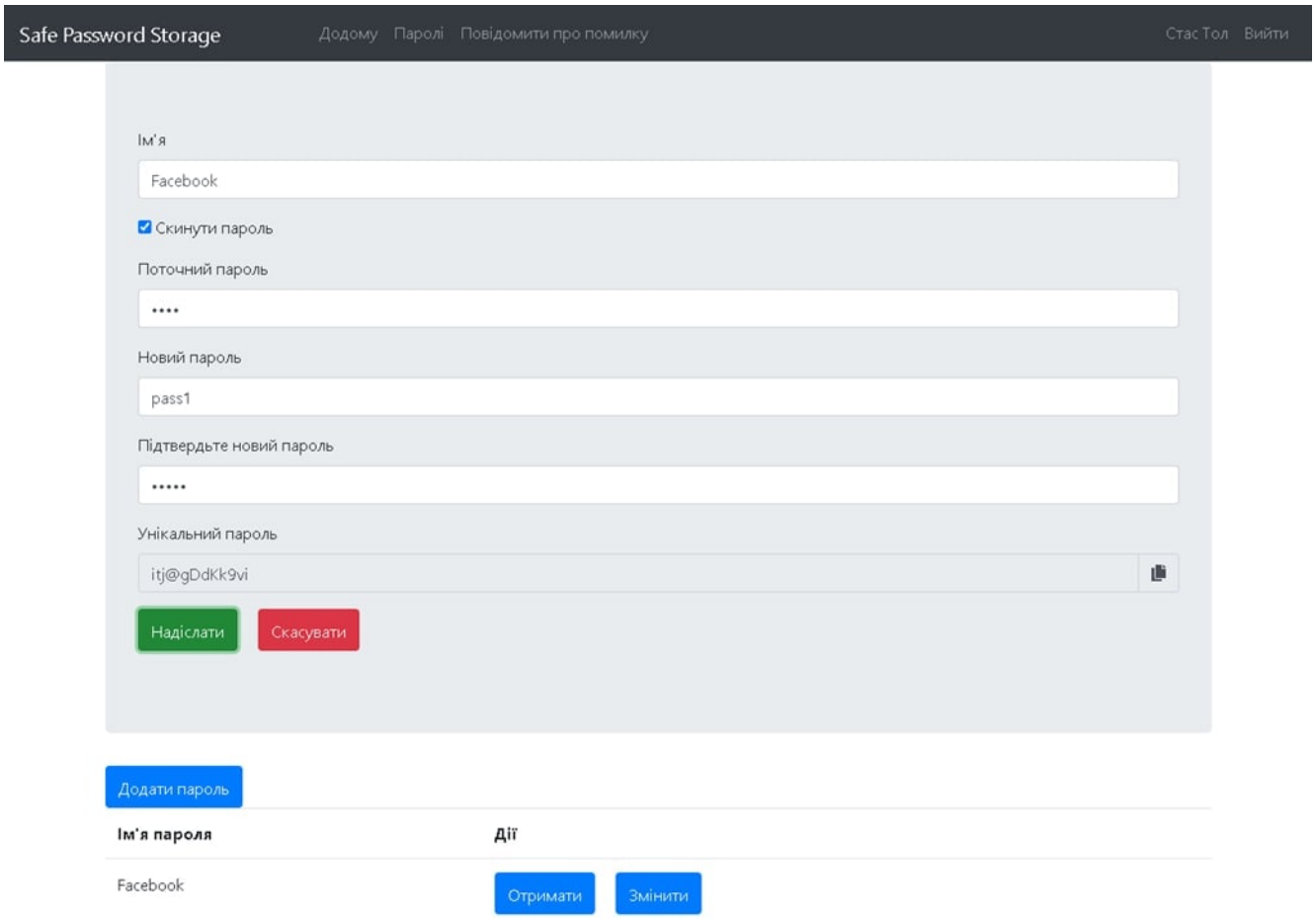


Рисунок 3.28 – Форма зміни існуючого пароля

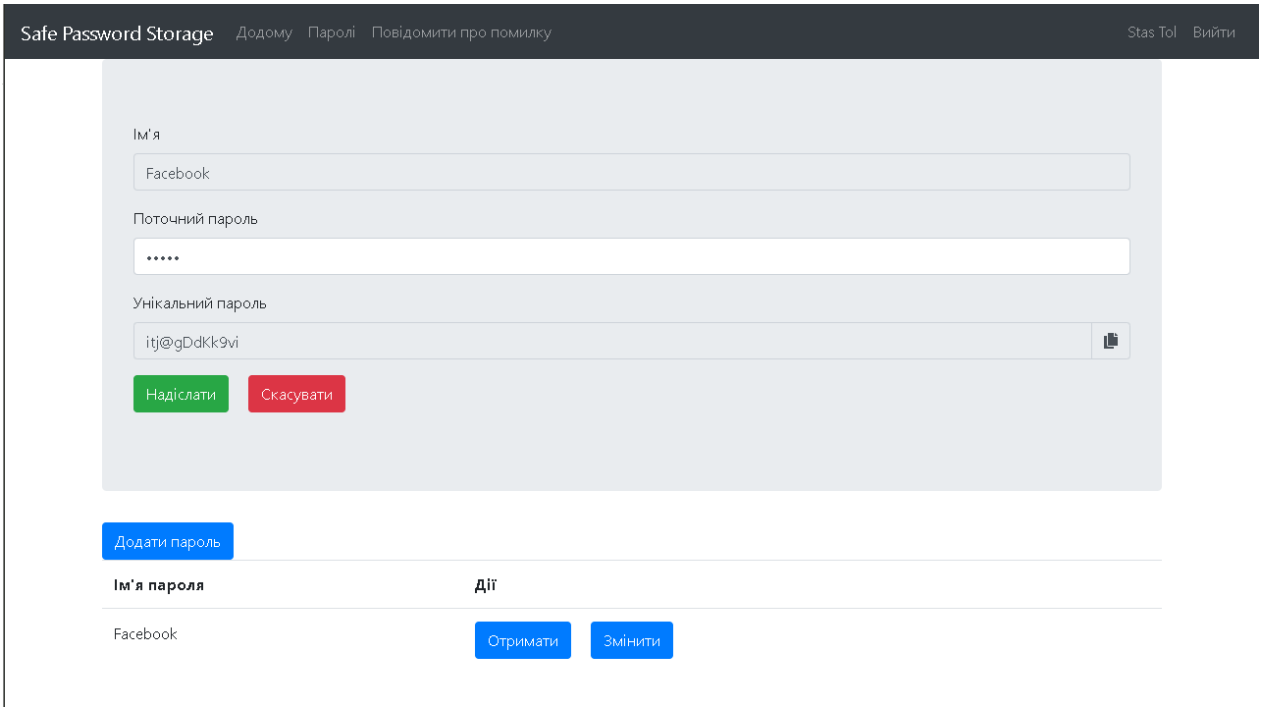


Рисунок 3.29 – Перевірка пароля після зміни

Якщо користувач введе звичайний пароль 5 раз невірно то система автоматично його заблокує (рис. 3.30).

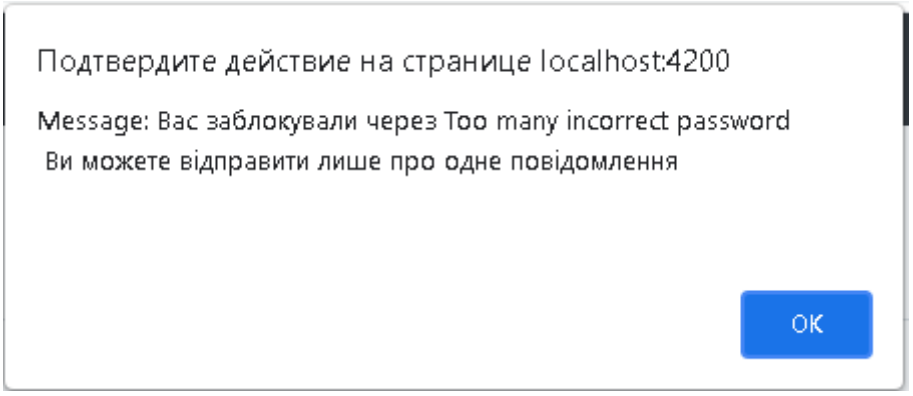


Рисунок 3.30 – Повідомлення про блокування користувача

Заблокований користувач має змогу відправити лише одне повідомлення про блокування адміністратору (рис. 3.31 – 3.34).

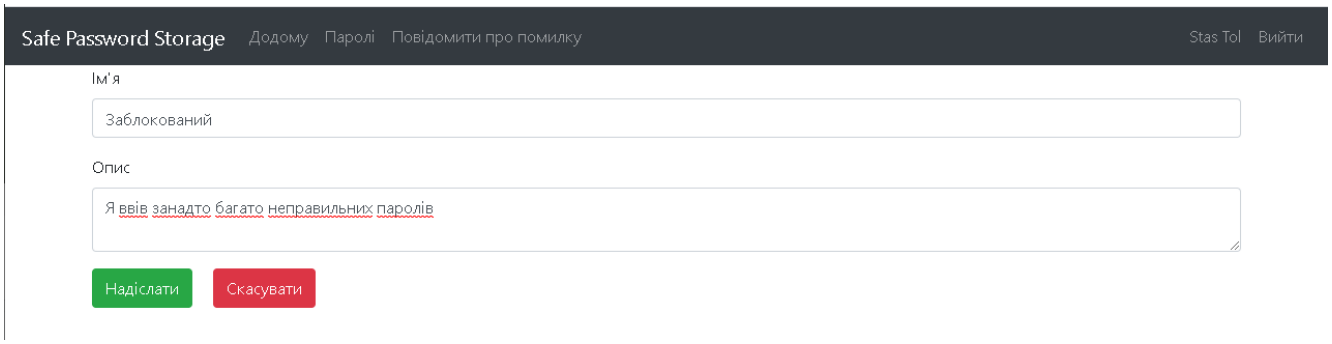


Рисунок 3.31 – Форма зворотного зв’язку

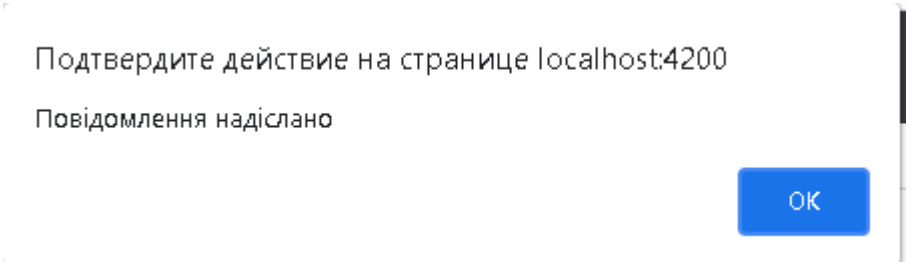


Рисунок 3.32 – Повідомлення про відправку повідомлення

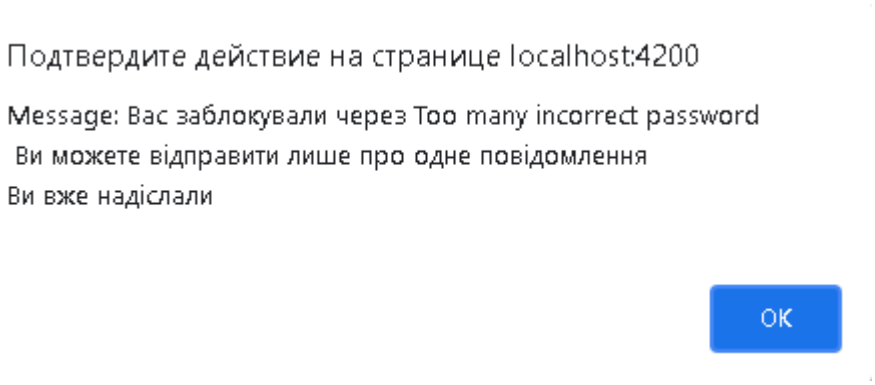


Рисунок 3.33 – Повідомлення про відмову на повторне відправлення

В меню сторінки адміністратора (рис. 3.34) відображаються переходи на сторінки зворотного зв’язку з користувачем (рис. 3.35) або системою (рис. 3.36) та список заблокованих користувачів (рис. 3.37).

Рисунок 3.34 – Меню сторінки адміністратора

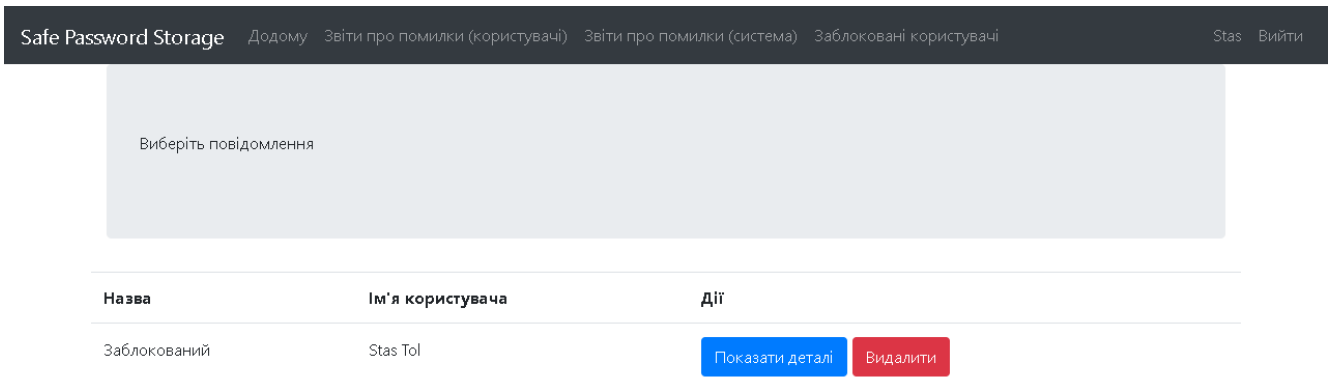


Рисунок 3.35 – Сторінка зворотного зв’язку з користувачем

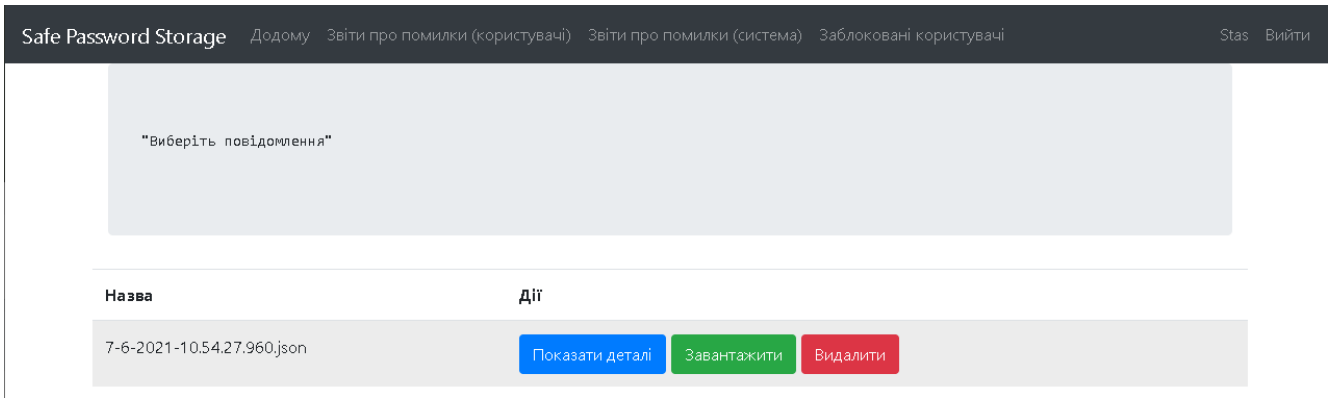


Рисунок 3.36 – Сторінка зворотного зв’язку з системою

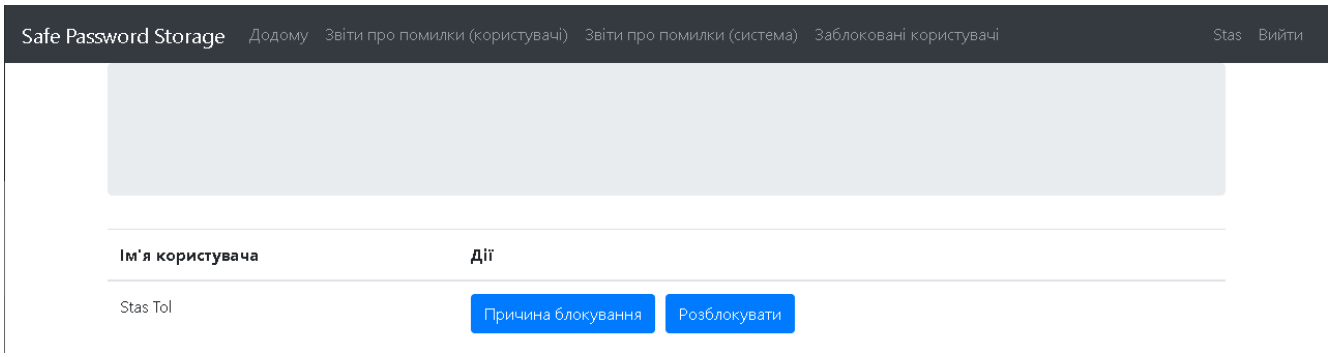


Рисунок 3.37 – Сторінка списку заблокованих користувачів

На сторінці зворотного зв’язку з користувачем адміністратор має змогу переглянути детальну інформацію повідомлення від користувача (рис. 3.38) та видалити його (рис. 3.39).

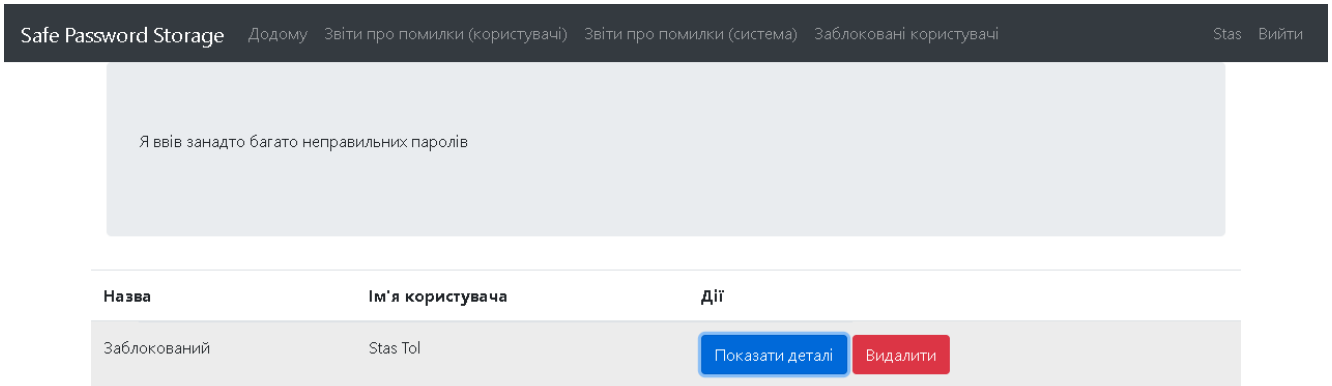


Рисунок 3.38 – Детальна інформація повідомлення

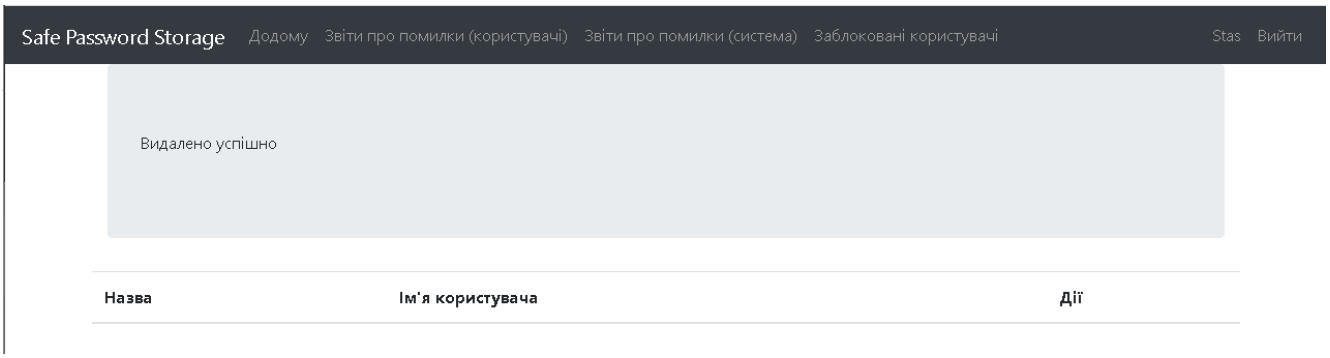


Рисунок 3.39 – Результат після видалення повідомлення

Сторінка зворотного зв’язку з системою дає змогу адміністратору переглянути детальну інформацію про помилку (рис. 3.40), завантажити цей файл (рис. 3.41) та видалити його (рис. 3.42).

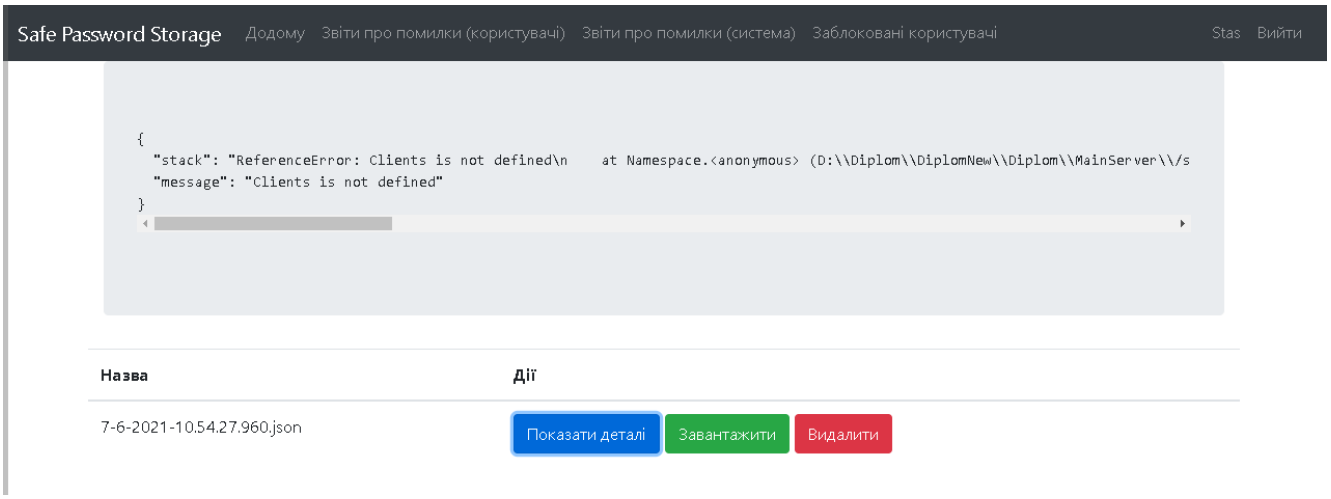


Рисунок 3.40 – Детальна інформація повідомлення про помилки в системі

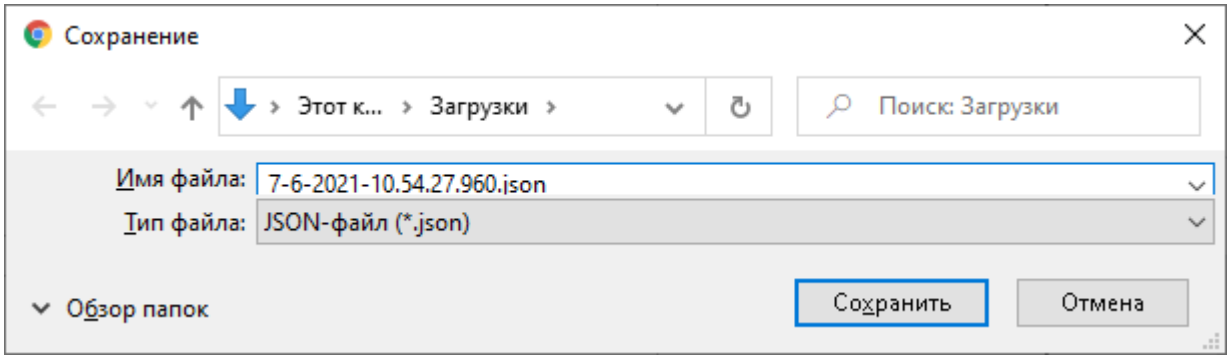


Рисунок 3.41 – Завантаження файлу

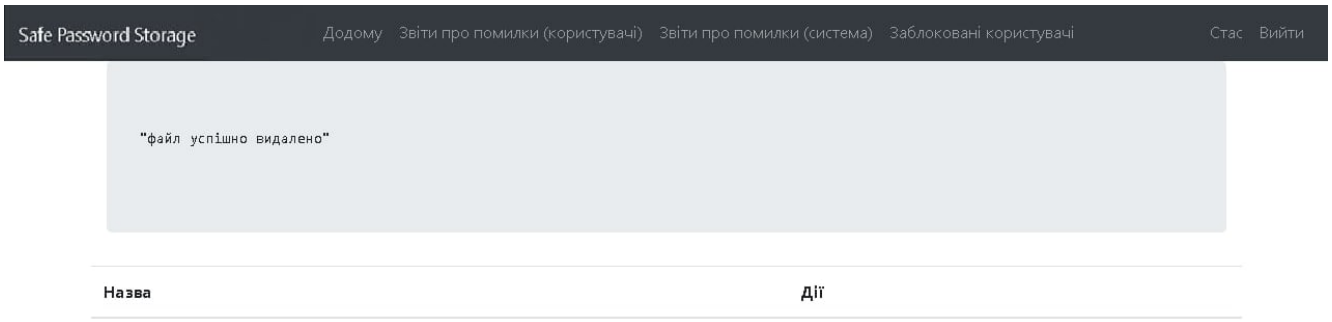


Рисунок 3.42 – Результат після видалення файлу

На сторінці з відображенням списку заблокованих користувачів адміністратор має змогу переглянути детальну інформацію про причину блокування користувача (рис. 3.43), а також зняти обмеження блокування з користувача (рис. 3.44-3.45).

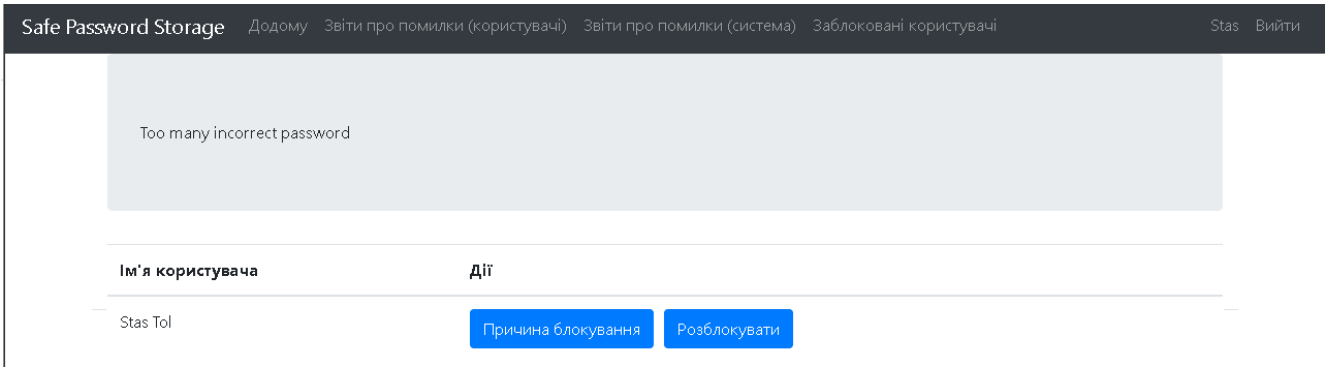


Рисунок 3.43 – Детальна інформація про заблокованого користувача

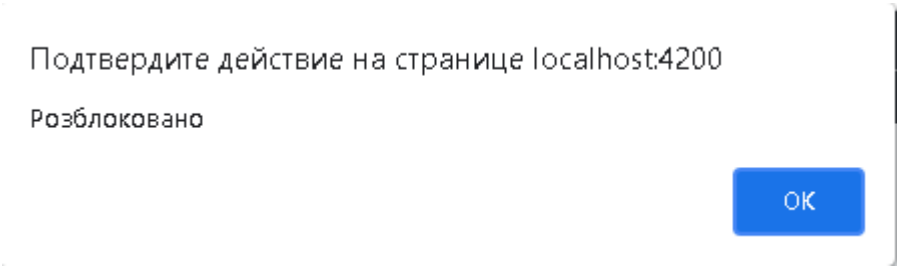


Рисунок 3.44 – Повідомлення про зняття обмеження з користувача

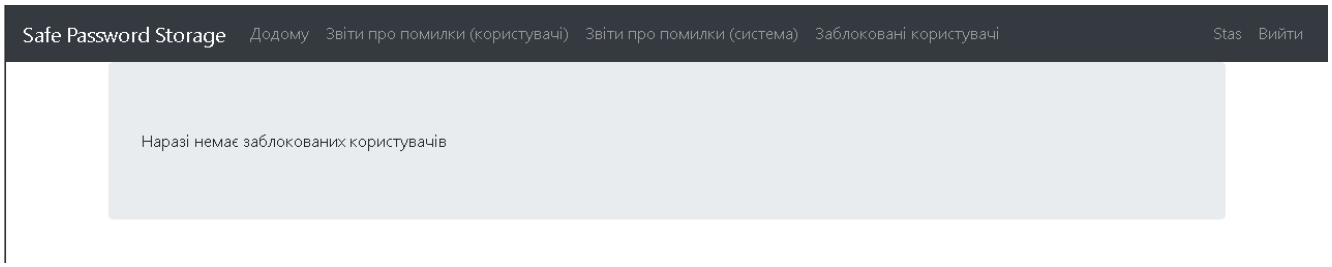


Рисунок 3.45 – Результат

На даний момент даний проект реалізовано у вигляді web-додатку. Він оптимізований під мобільні пристрої (рис. 3.46).

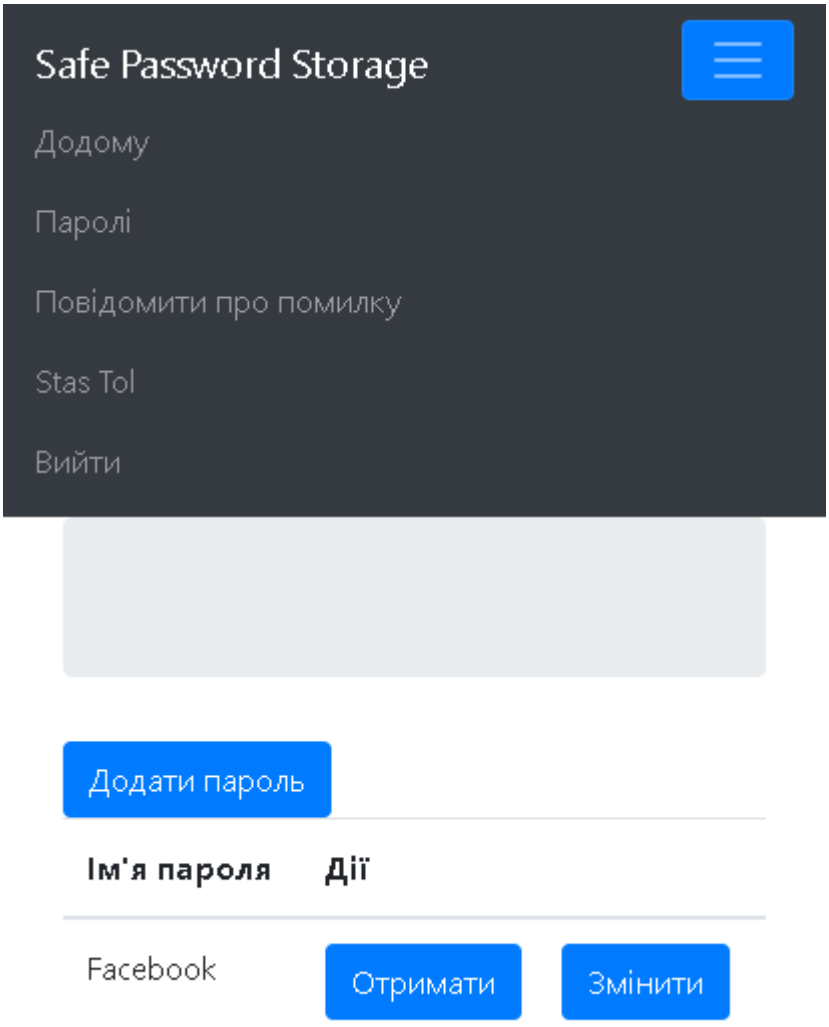


Рисунок 3.46 – Вигляд web-сторінки на мобільному пристрою

Всі запити на сервер даного проекту були протестовані за допомогою Postman (рис. 3.47), приклад тесту авторизації зображено на рисунку 3.48. Він дозволив перевірити вірність обробки сервером значень, що були відправлені під час запитів.

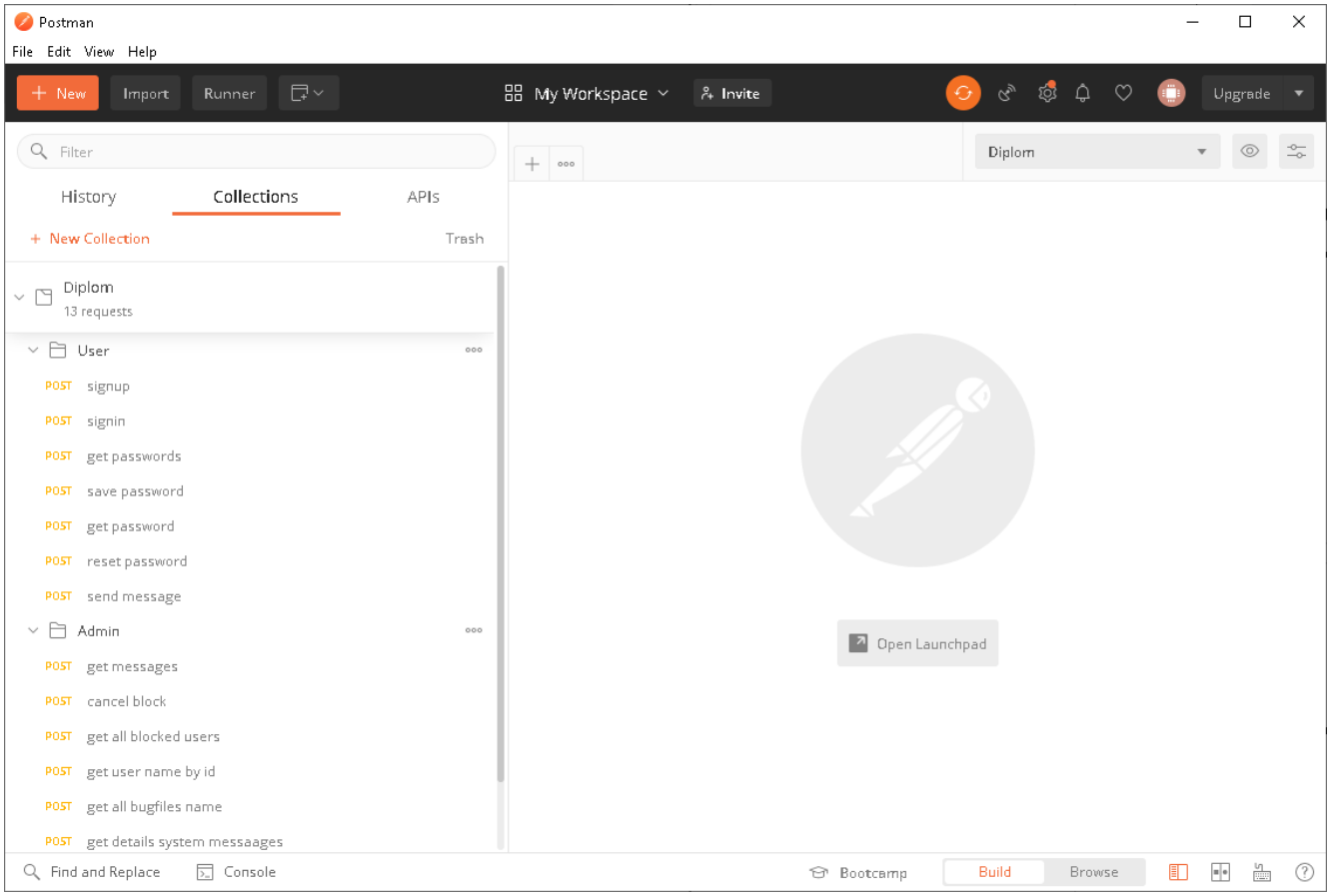


Рисунок 3.47 – Програма Postman

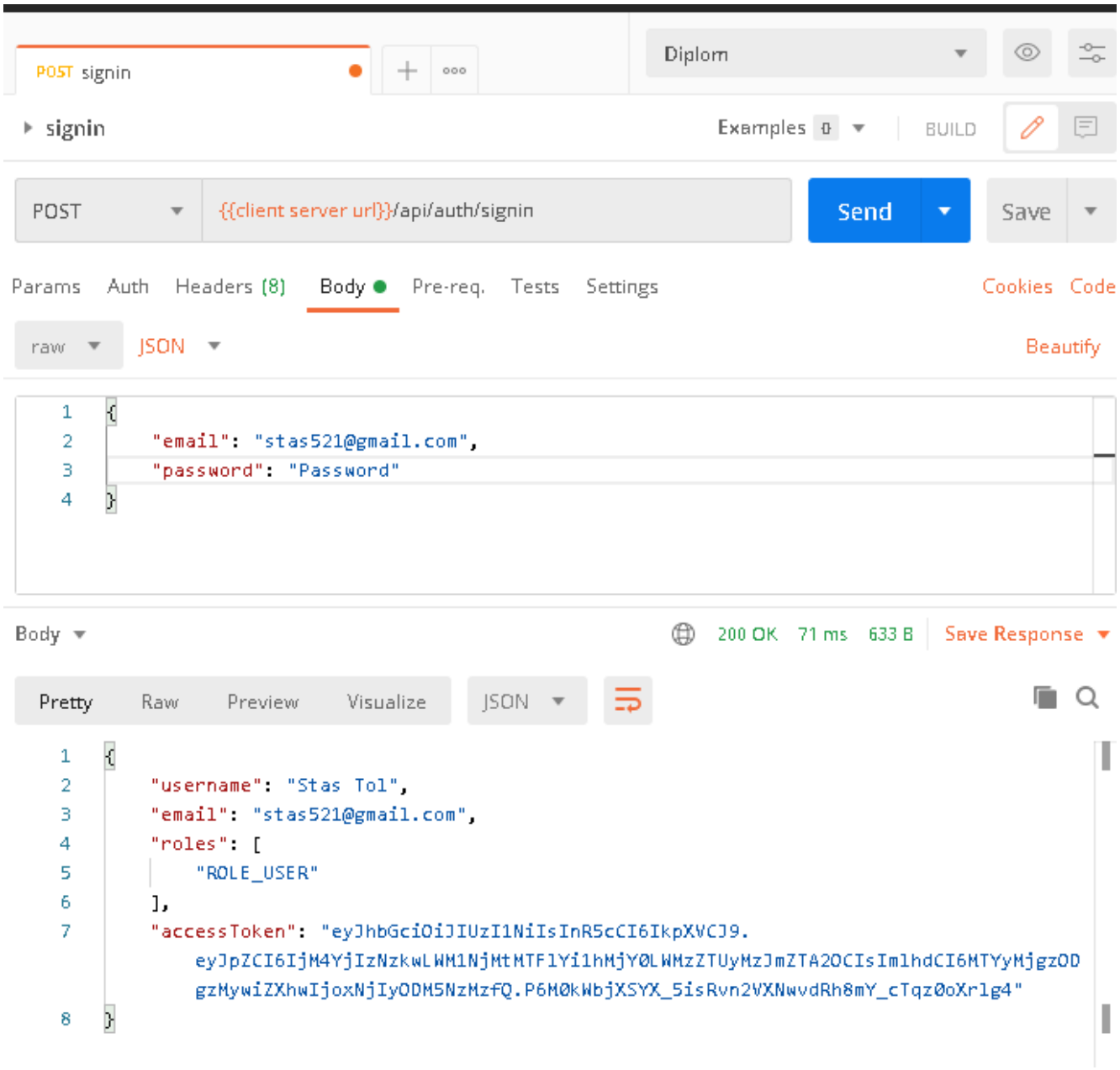


Рисунок 3.48 – Приклад тестування запиту на авторизацію

Декілька паролів були перевірені на складність взлому на сайті https://www.my1login.com/resources/password-strength-test/ і отримали високу оцінку надійності (рис. 3.49).

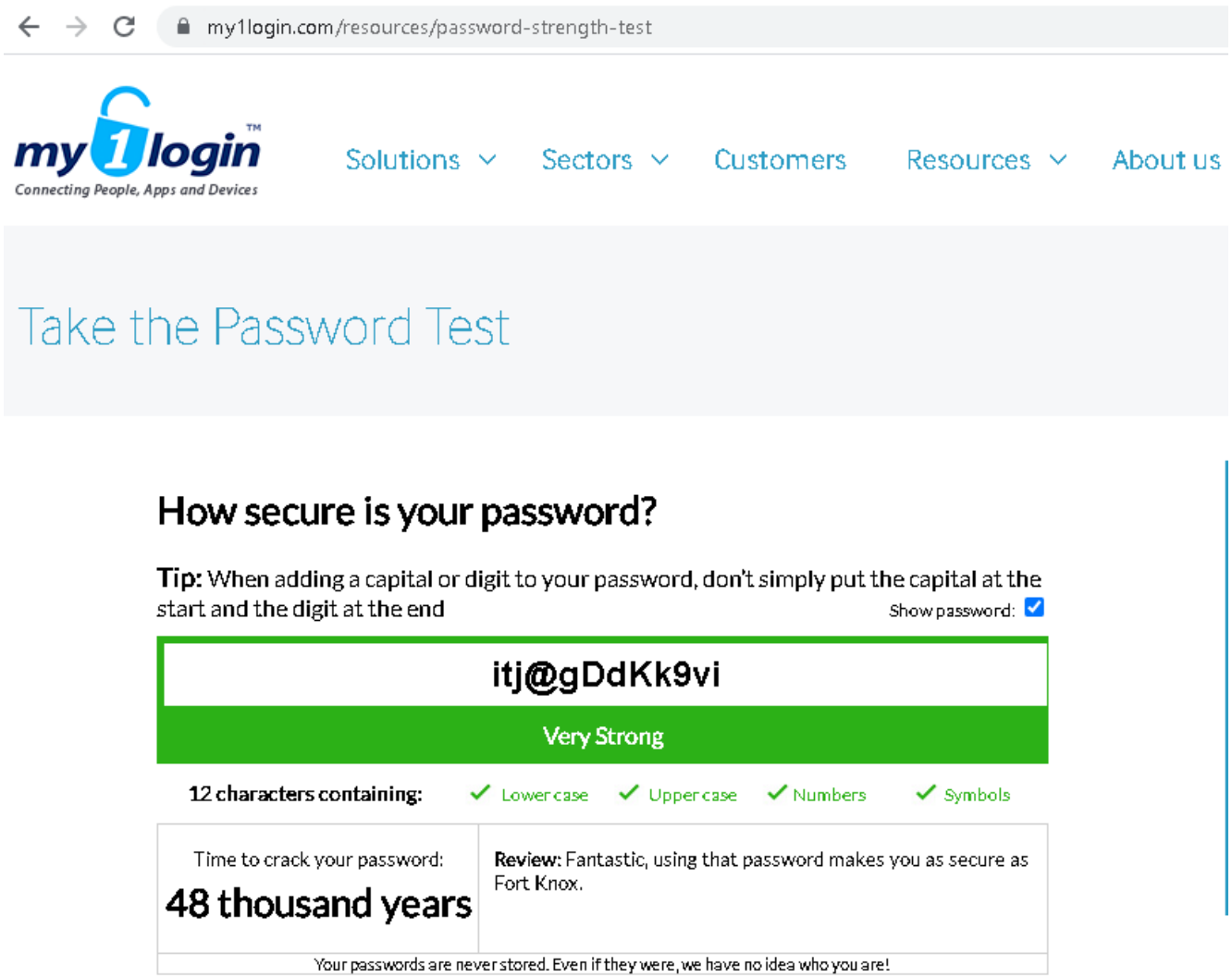


Рисунок 3.49 – Приклад тестування паролю

ВИСНОВОК

Під час проектування кваліфікаційної роботи бакалавра було проведено аналіз наявних аналогічних рішень на ринку, розглянуті їх сильні та слабкі сторони. Майже відсутність менеджерів для унікального збереження паролів в базі даних змусила розробити власну web-додаток.

Для реалізації продукту проекту з функцією не збереження паролів в базі даних на стороні клієнта була обраний фреймворк Angular, а серверна частина складається з двох серверів, що реалізовані за допомогою фреймворк Node.js. Для вирішення даної задачі з використанням баз даних було створено два сервери:

– перший – клієнтський сервер з використанням бази даних MySQL;
– другий – головний для збереження хеш-кодів.

Для роботи головного сервера було використано базу даних mongoDB. MongoDB не вимагає опису схеми таблиць в порівнянні з реляційними базами даних. Особливістю його роботи є те, що дані зберігаються у вигляді колекцій де структура даних та документів не обов'язково повинна бути схожа.

Роботу серверів реалізовано у вигляді REST API, та забезпечено синхронізацію з базою даних. Для авторизації користувача було використано токен і додано адміністративний інтерфейс, що дало змогу створювати функціонал для будь-якої системи. В процесі роботи також було використано протокол HTTPS для захисту від MITM атак.

Одним з головних етапів створення проекту “Програма генерації та збереження паролів “Safe Password Storage” є його моделювання за допомогою мови моделювання UML. Розроблення та побудова різних видів діаграм допомогло створити етапи розробки програми з отриманням уявлення про кінцевий результат додатку.

Створення програми генерації та збереження паролів “Safe Password Storage” на сьогодні має важливе значення. Захист конфіденційної інформації користувачів – один з пріоритетів розвитку IT-технології сучасного світу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Оригинальный способ генерации мастер-пароля [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/vdsina/blog/516414/

2. Найважливіші недоліки безпеки, знайдені в LastPass у Chrome [Електронний ресурс] – Режим доступу до ресурсу: https://uk.allsoftfree.com/critical-security-flaws-found-lastpass-chrome-33913

3. Revisiting Security Vulnerabilities in Commercial Password Managers [Електронний ресурс] – Режим доступу до ресурсу: http://eprints.whiterose.ac.uk/158056/8/Revisiting_Security_Vulnerabilities_in_Commercial_Password_Managers_2.pdf

4. Плюси і мінуси Dashlane [Електронний ресурс] – Режим доступу до ресурсу: https://sravni.cc/reviews/plyusy-i-minusy-programmy-dashlane/

5. Плюси і мінуси KeePass [Електронний ресурс] – Режим доступу до ресурсу: https://sravni.cc/reviews/plyusy-i-minusy-keepass-kipas/

6. МЕТОДОЛОГІЯ IDEF0 [Електронний ресурс] – Режим доступу до ресурсу: https://stud.com.ua/87184/ekonomika/metodologiya_idef0 .

7. Діаграма класів [Електронний ресурс] – Режим доступу до ресурсу: https://uk.photo-555.com/7917237-class-diagram .

8. Діаграма діяльності UML [Електронний ресурс] – Режим доступу до ресурсу: http://pingwin.lg.ua/post/diagrama/uk/deatelnost-diagrama-dialnosti-uml-planerka-onlajn-skola-kreativnosti.htm

9. UML [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-uml-models

10. Діаграми потоків даних- DFD [Електронний ресурс] – Режим доступу до ресурсу: https://thelib.info/tehnologii/3215921-diagrami-potokiv-danih-dfd/

11. Етапи проектування та моделювання БД [Електронний ресурс] – Режим доступу до ресурсу: http://ua.textreferat.com/referat-23369-3.html .

ДОДАТОК А

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку інформаційної системи
«Програма генерації та збереження паролів "Safe Password Storage"»

Суми 2021

1. Призначення й мета створення web-додатку

Призначення web-додатку

Узагальненою постановкою задачі проектування Web-додатку є створення програми за допомогою якої користувач матиме змогу створювати унікальні паролі, що даватимуть можливість авторизації на різних сайтах.

1.2 Мета створення web-додатку

Головною метою проекту є створення web-додатку, який допоможе користувачу використовувати один пароль для генерації унікальних паролів для входу в усі сервіси.

1.3 Цільова аудиторія

Програмний продукт розроблений для будь-якого користувача, що потребує захисту від хакерів та злому своїх акаунтів шляхом створення та зберігання унікальних паролів.

2 Вимоги до web-додатку

2.1 Вимоги до web-додатку вцілому

2.1.1 Вимоги до структури й функціонування web-додатку

Реалізація Web-додатку розроблена у вигляді сайту і має бути доступним в мережі Інтернет з доменним іменем tolstonozhenko.com.ua. Web-додаток повинен складатися із взаємозалежних розділів та чітко розділеними функціями.

2.1.2 Вимоги до користувача

Користувач повинен мати базові знання для роботи з персональним комп'ютером і стандартним web-браузером. Для використання web-додатку не вимагається володіти особливими технічними навичками.

Для роботи з додатком адміністратору достатньо мати базові навички в роботі з адмініструванням сайту.

2.1.3 Вимоги до збереження інформації

Інформація про користувачів буде зберігатися в базі даних MySQL а хеш-код - в базі даних MongoDB.

2.1.4 Вимоги до розмежування доступу

Інформація в web-додатку має бути загальнодоступною.

Відповідно до прав доступу користувачі діляться на дві групи:

1. клієнт;
2. адміністратор;

Клієнти мають право створювати та змінювати власні паролі.

Адміністратор працює з клієнтом та, при отриманні відповідного повідомлення, допомагає вирішити проблеми, з якими може зіткнутися клієнт в додатку: блокування користувача або припинення роботи додатку. При випадковому блокуванні користувача додатком, адміністратор має право його розблокувати.

2.2 Структура web-додатку

2.2.1 Загальна інформація про структуру web-додатку

Web-додаток має такі розділи:

- Головна сторінка - загальна інформація про web-додаток.
- Вікно авторизації - для авторизації користувача.
- Вікно реєстрації - для реєстрації користувача.

- Особистий кабінет користувача.
- Вікно списків паролів – містить список створених паролів користувача.
- Вікно створення паролю – для створення нового пароля.
- Вікно пароля – для введення «звичайного» пароля і отримання пароля для входу.

2.2.2 Навігація

Web-додаток буде забезпечувати доступ до всіх можливих розділів користувача з відображенням відповідної інформації.

2.2.3 Наповнення web-додатку (контент)

Управління контентом web-додатку буде відбуватися за допомогою хостинга Hostpro.

Заповнення та редагування контенту web-додатку може тільки користувач, який має доступ до менеджерів файлів домену tolstonozhenko.com.ua сайту Hostpro.

Доступ до входу менеджера файлів буде надавати головний користувач домену tolstonozhenko.com.ua за допомогою створення облікового запису FTP.

2.2.4 Дизайн та структура додатку

Стиль web-додатку має бути сучасним, робочим, мати приємний дизайн. Структура або меню додатку - доступною та зрозумілою для звичайного користувача. У якості основних кольорів запропоновано використати сірий та білі відтінки.

Дизайн сторінки повинен бути сумісним з мобільними пристроями.

Схема елементів на головні сторінці web-додатку матиме вигляд, як показано на рис. 2.1 та 2.2.

Рисунок 2.1 – Схема головної сторінки для персональних комп’ютерів

Рисунок 2.2 – Схема головної сторінки для мобільних пристроїв

2.2.5 Система навігації (карта web-додатку)

Карта web-додатку зображена на рис. 2.3.

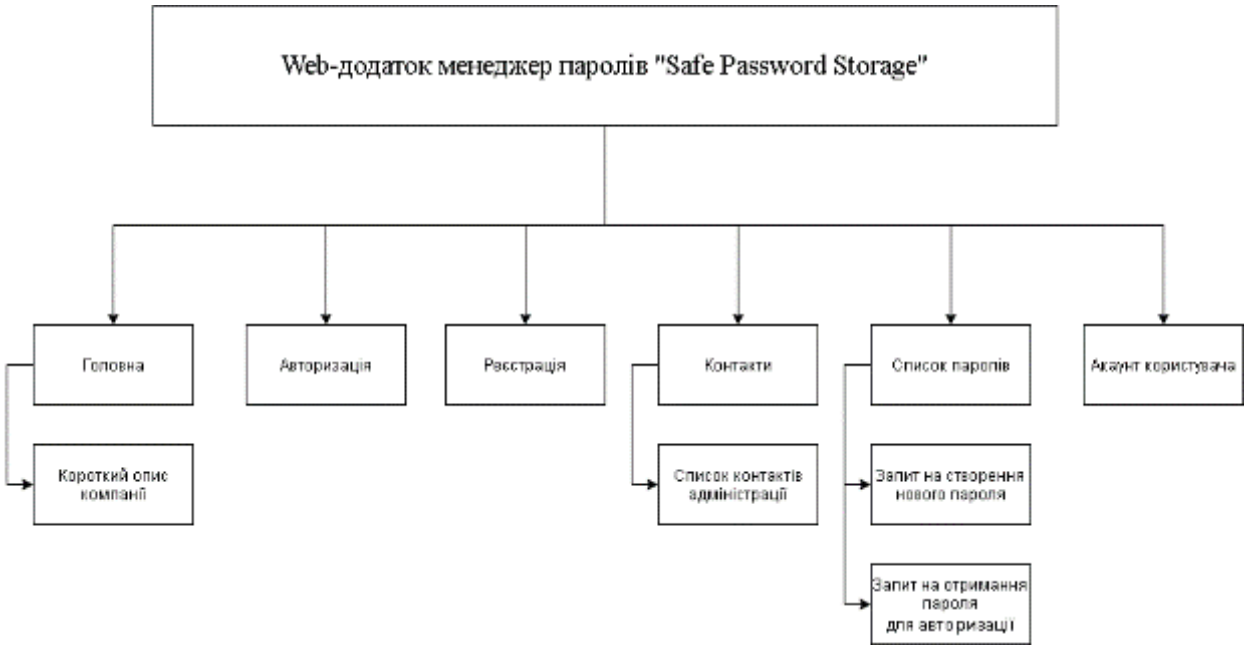


Рисунок 2.3 – карта web-додатку

2.3 Вимоги до функціонування системи

2.3.1 Потреби користувача

Перелік потреб користувача при роботі з web-додатком наведено у таблиці 2.1.

Таблиця 2.1 – Потреби користувача

ID	Потреби користувача	Джерело
UN-01	Запит на створення нового пароля	Клієнт
UN-02	Запит на отримання пароля для авторизації	Клієнт
UN-03	Розблокування користувача	Адміністратор
UN-04	Можливість зворотного зв’язку	Клієнт
UN-05	Перегляд інформації про компанію	Клієнт

2.3.2 Функціональні вимоги

Основні функціональні вимоги з урахуванням потреб користувачів:

– реєстрація та авторизація користувачів у web-додатку;

- можливість створення пароля авторизованими користувачами;
- можливість отримання пароля для авторизації на інших сайтах авторизованими користувачами;
- розблокування користувача адміністратором;
- можливість зміни або додавання методів створення паролів адміністратором.

2.3.3 Системні вимоги

Даний розділ містить основні системні вимоги, визначені розробником. Перелік систем наведений в таблиці 2.2.

Таблиця 2.2 – Системні вимоги

ID	Системні вимоги	Пріоритет	Опис
SR-01	Наявність модуля авторизації	M	Надає можливість клієнту провести авторизацію та реєстрацію
SR-02	Наявність модуля захисту від втручання в систему	M	Блокує користувача якщо він декілька раз вводить невірні дані
SR-03	База даних користувачів	M	Зберігає інформацію про користувачів в базі даних
SR-04	База даних хеш-кодів	M	Зберігає інформацію з хеш-кодами в базі даних
SR-05	Модуль взаємодії між двома серверами	M	Проводить з’єднання між серверами за допомогою сертифікатів

Продовження таблиці 2.2

SR-06	Модуль взаємодії між сервером та клієнтом	M	Робить з’єднання між сервером та клієнтом за допомогою токена та сертифіката
SR-07	Наявність модуля зворотного зв’язку	S	Надає можливість клієнту зв’язатися з адміністратором
SR-08	Сторінка адміністратора	M	Надає можливість для розблокування користувача адміністратором
SR-09	Модуль додавання та редагування методів створення паролів	C	Відповідає за можливість додати новий спосіб створення пароля

Умовні позначення в таблиці А.2:

Must have (M) – вимоги, які повинні бути реалізовані в системі;

Should have (S) – вимоги, які мають бути виконані, але вони можуть почекати своєї черги;

Could have (C) – вимоги, які можуть бути реалізовані, але вони не є головною ціллю проекту.

2.4 Вимоги до видів забезпечення

2.4.1 Вимоги до інформаційного забезпечення

Реалізація web-додатку відбувається з використанням:

- Node.js 14.16.0
- Angular 7.2.11
- MongoDB 4.4.4
- MySQL 8.0

2.4.2 Вимоги до лінгвістичного забезпечення

Web-додаток може бути як українською так і англійською мовами.

2.4.3 Вимоги до програмного забезпечення

Програмне забезпечення клієнтської частини повинне задовольняти наступним вимогам:

- Web-браузер: Internet Explorer 7.0 і вище, або Firefox 3.5 і вище, або Opera 9.5 і вище, або Safari 3.2.1 і вище, або Chrome 2 і вище.

3 Склад і зміст робіт зі створення web-додатку

Докладна послідовність створення web-додатку наведено в таблиці 3.1.

Таблиця 3.1 – Етапи створення web-додатку

№	Склад і зміст робіт	Строк розробки (у робочих днях)
1	Інціалізація проекту	4 дні
2	Планування проекту	2 дні
3	Створення клієнтського сервера	10 дні
4	Створення файлового сервера	10 дні
5	Створення макету дизайну web-додатку	1 день
6	Верстка	3 дні
7	Робота з контентом	1 день
8	Тестування проекту	3 дні
9	Перевірка працездатності web-додатку	1 день
10	Завершення роботи	1 день
	Загальна тривалість робіт	36 днів

4 Вимоги до складу й змісту робіт із введення web-додатку в експлуатацію

Для використання web-додатку необхідно розмістити його у мережі Інтернет. Розробник повинен мати доменне ім’я та місце на хостингу: tolstonozhenko.com.ua та Hostpro відповідно. Хостинг має містити в собі web-додаток і базу даних MySQL. База даних MongoDB буде зберігатися на їхньому сервері.

Web-додаток буде містити два сервера «основний» та «клієнтський», які зв’язані між собою за допомогою сертифікатів. Доступ до «основного» сервера буде мати лише «клієнтський» сервер. Передача даних між користувачем та клієнтським сервером відбувається за допомогою відкритих та закритих ключів.

ДОДАТОК Б

ПЛАНУВАННЯ РОБІТ

на розробку інформаційної системи
«Програма генерації та збереження паролів "Safe Password Storage"»

Суми 2021

1. ОПИСАННЯ ІТ-ПРОЕКТУ НА ФАЗІ РОЗРОБЛЕННЯ

1.1 Планування змісту робіт

Для зручності виконання роботи та прискорення робочого процесу завдання створення проекту доцільно поділити на більш дрібні і керовані частини. Для цієї цілі використовують інструмент WBS.

WBS використовує структуру декомпозиції, яка охоплює всі кроки створення проекту для ділення на легко керовані етапи. Вони, в свою чергу, поділяються до можливості для створення найлегшого етапу.

WBS-структура даного проекту наведена на рис. 2.1.

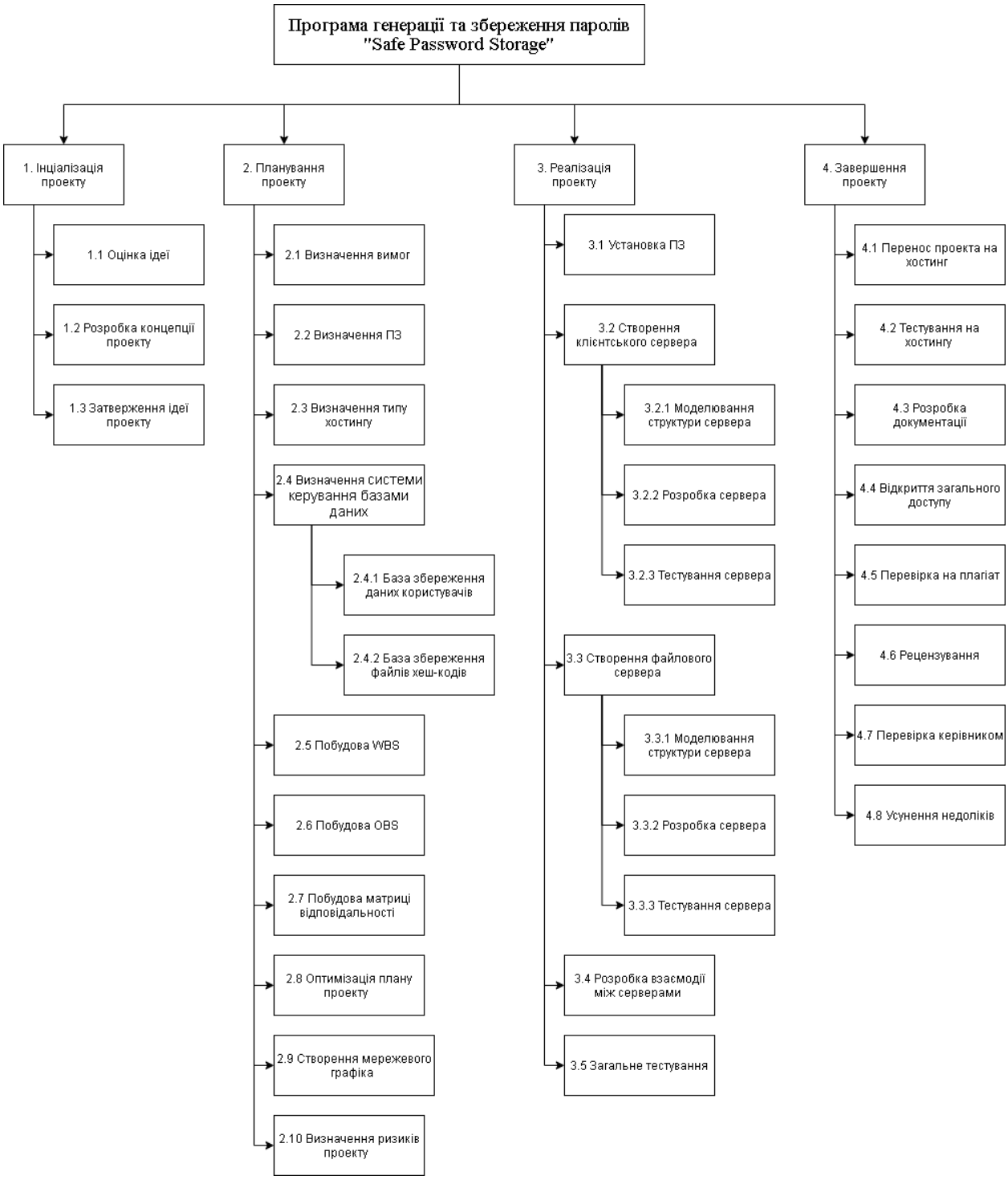


Рисунок 2.1 – WBS-структура проекту

1.2 Планування структури виконавця для впровадження готового проекту

Після побудови WBS відбувається розробка організаційної структури виконавців. До OBS входять всі учасники з урахуванням їх ролей і відносин підлеглості, що приймають участь у створенні проекту. Далі визначаються виковці за переліком етапів робіт нижнього рівня кожної гілки WBS-структури.

Список виконавців проекту представлено в таблиці 2.1.

Таблиця 2.1 – Виконавці проекту

Роль	Ім’я	Проектна роль
Програміст	Толстоноженко С.О.	Розробляє та виконує тестування основного функціоналу проекту.
Дипломний керівник	Шендрик В.В.	Відповідає за виконання термінів, виконує збір та аналіз даних.
Організаційний менеджер	Толстоноженко С.О.	Відповідає за створення документації і планів виконання.

OBS-структура проекту зображена на рис. 2.2.

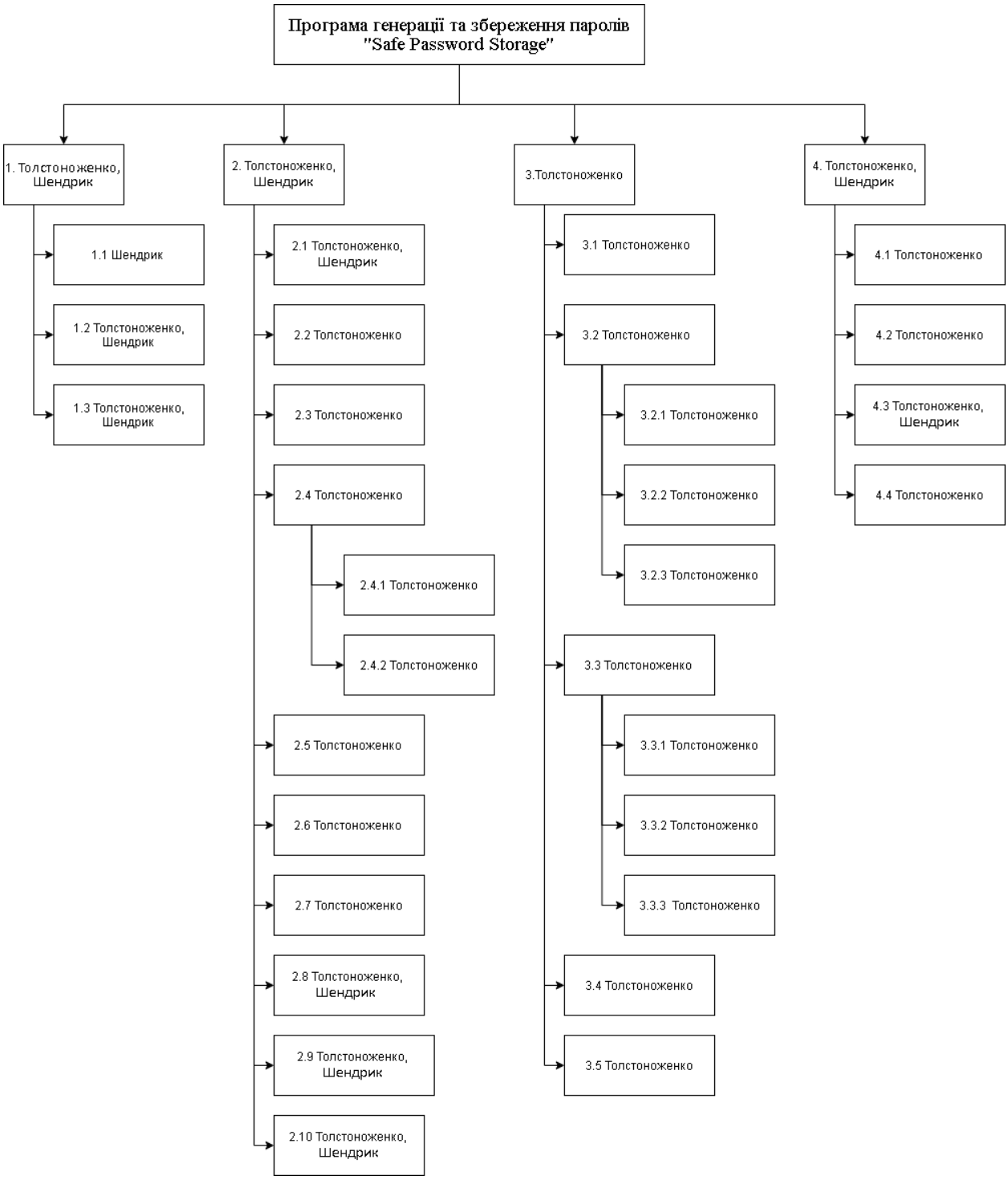


Рисунок 2.2 – OBS - структура проекту

1.3 Побудова матриці відповідальності

Побудова матриці відповідальності дає можливість назначити відповідального виконавця на кожному етапі розробки проекту, будується на основі WBS і OBS, дає можливість контролювати відповідність розподілу ролей цілям проекту.

Матриця відповідальності проекту наведено на рисунку 2.3.

WBS \ OBS			Толстоноженко	Шендрик
1	1.1			О
	1.2		О	У
	1.3		У	О
2	2.1		О	У
	2.2		О	
	2.3		О	
	2.4	2.4.1	О	
		2.4.2	О	
	2.5		О	
	2.6		О	
	2.7		О	
	2.8		О	У
	2.9		О	У
	2.10		О	У
3	3.1		О	
	3.2	3.2.1	О	
		3.2.2	О	
		3.2.3	О	
	3.3	3.3.1	О	
		3.3.2	О	
		3.3.3	О	
	3.4		О	
4	4.1		О	
	4.2		О	
	4.3		О	У
	4.4		О	
	4.5		У	О
	4.6		О	
	4.7			О
	4.8		О	
О-основна робота; У-приймає участь				

Рисунок 2.3 - Матриця відповідальності

1.4 Розробка PDM мережі

Створення проекту займає тривалий час. Щоб уникнути виникнення проблем з дедлайном потрібно побудувати PDM мережу. В ній представлено діаграму, яка за допомогою стрілок відображає послідовність виконання запланованих задач.

Щоб побудувати PDM мережу використано Microsoft Project. Результат мережі зображено на рисунках 2.4-2.6.

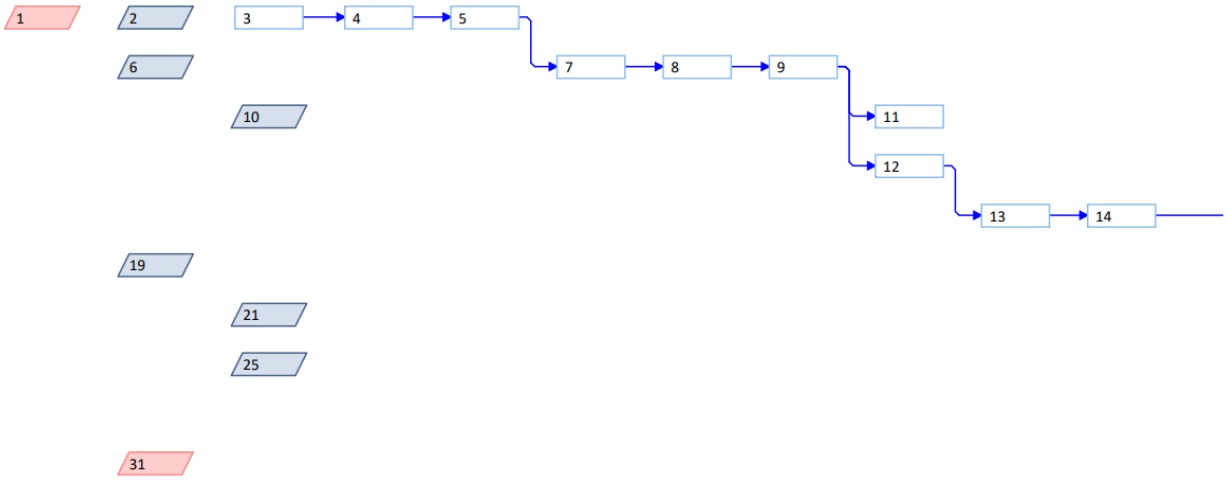


Рисунок 2.4 - PDM мережа

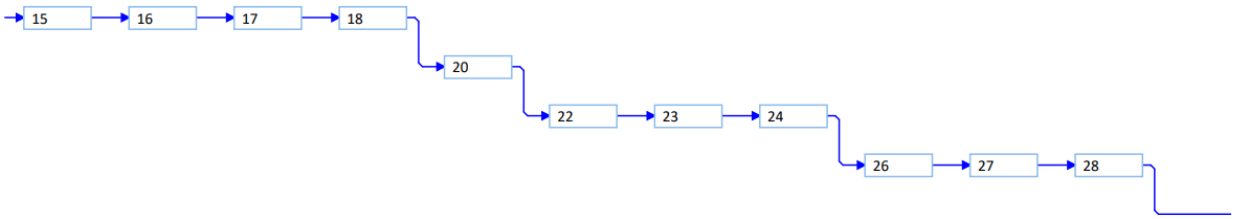


Рисунок 2.5 - продовження PDM мережі

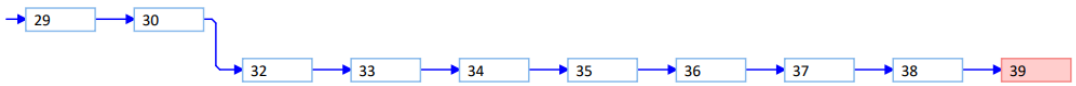
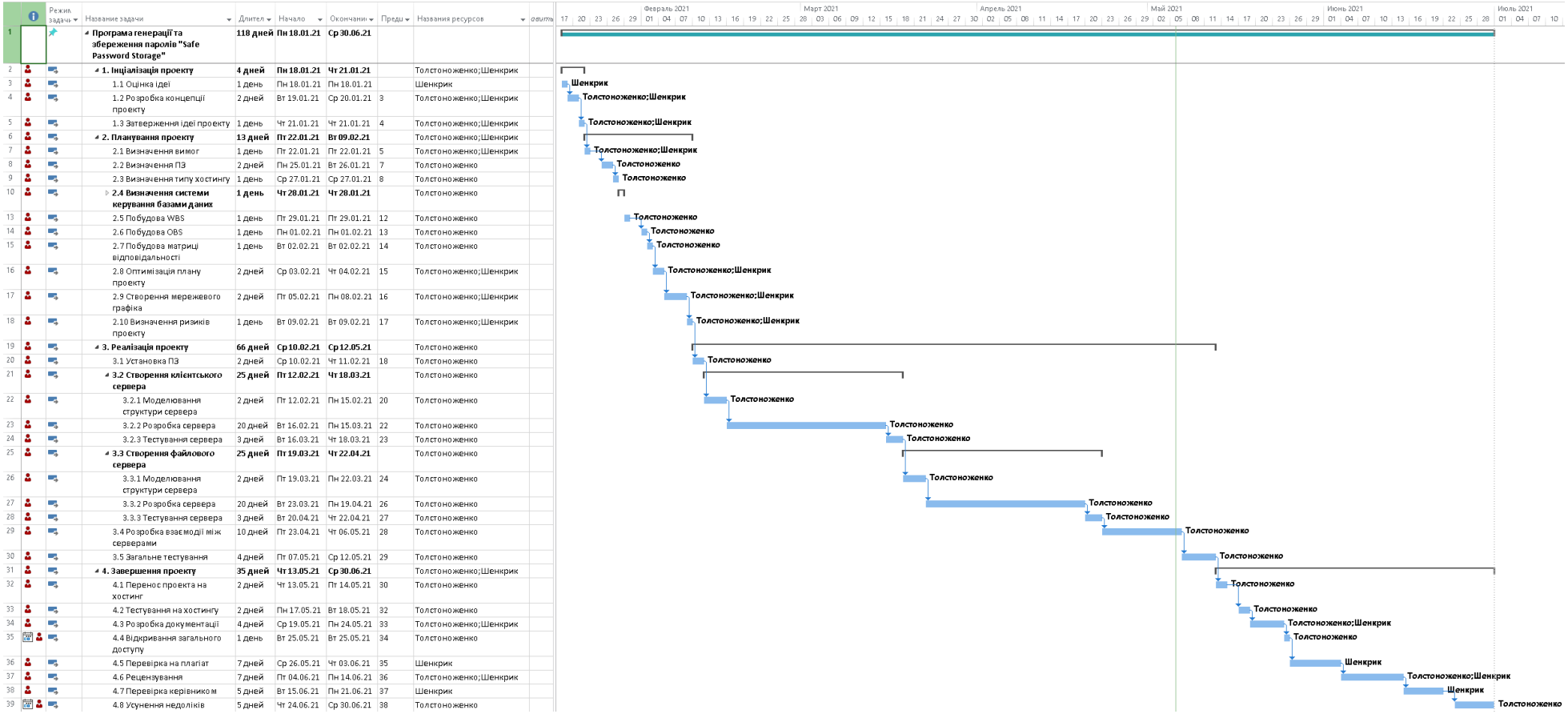


Рисунок 2.6 - продовження PDM мережі

1.5 Побудова календарного графіку

Календарне планування - це процес складання розкладу, строків виконання розробки проекту з урахуванням різних видів матеріально-технічних та трудових ресурсів. Розклад відображає планові та фактичні дані про початок, тривалість і кінець створення проекту, взявши за основу робочий елемент WBS.

Календарний графік проекту представлений у вигляді діаграми Ганта і наведений на рисунку 2.7.



	Режим задачи	Название задачи	Длитель	Начало	Окончани	Преду	Названия ресурсов
1		**Програма генерації та збереження паролів "Safe Password Storage"**	**118 дней**	**Пн 18.01.21**	**Ср 30.06.21**		
2		**1. Ініціалізація проекту**	**4 дней**	**Пн 18.01.21**	**Чт 21.01.21**		Толстоноженко;Шенкрик
3		1.1 Оцінка ідеї	1 день	Пн 18.01.21	Пн 18.01.21		Шенкрик
4		1.2 Розробка концепції проекту	2 дней	Вт 19.01.21	Ср 20.01.21	3	Толстоноженко;Шенкрик
5		1.3 Затвердження ідеї проекту	1 день	Чт 21.01.21	Чт 21.01.21	4	Толстоноженко;Шенкрик
6		**2. Планування проекту**	**13 дней**	**Пт 22.01.21**	**Вт 09.02.21**		Толстоноженко;Шенкрик
7		2.1 Визначення вимог	1 день	Пт 22.01.21	Пт 22.01.21	5	Толстоноженко;Шенкрик
8		2.2 Визначення ПЗ	2 дней	Пн 25.01.21	Вт 26.01.21	7	Толстоноженко
9		2.3 Визначення типу хостингу	1 день	Ср 27.01.21	Ср 27.01.21	8	Толстоноженко
10		**2.4 Визначення системи керування базами даних**	**1 день**	**Чт 28.01.21**	**Чт 28.01.21**		Толстоноженко
13		2.5 Побудова WBS	1 день	Пт 29.01.21	Пт 29.01.21	12	Толстоноженко
14		2.6 Побудова OBS	1 день	Пн 01.02.21	Пн 01.02.21	13	Толстоноженко
15		2.7 Побудова матриці відповідальності	1 день	Вт 02.02.21	Вт 02.02.21	14	Толстоноженко
16		2.8 Оптимізація плану проекту	2 дней	Ср 03.02.21	Чт 04.02.21	15	Толстоноженко;Шенкрик
17		2.9 Створення мережевого графіка	2 дней	Пт 05.02.21	Пн 08.02.21	16	Толстоноженко;Шенкрик
18		2.10 Визначення ризиків проекту	1 день	Вт 09.02.21	Вт 09.02.21	17	Толстоноженко;Шенкрик
19		**3. Реалізація проекту**	**66 дней**	**Ср 10.02.21**	**Ср 12.05.21**		Толстоноженко
20		3.1 Установка ПЗ	2 дней	Ср 10.02.21	Чт 11.02.21	18	Толстоноженко
21		**3.2 Створення клієнтського сервера**	**25 дней**	**Пт 12.02.21**	**Чт 18.03.21**		Толстоноженко
22		3.2.1 Моделювання структури сервера	2 дней	Пт 12.02.21	Пн 15.02.21	20	Толстоноженко
23		3.2.2 Розробка сервера	20 дней	Вт 16.02.21	Пн 15.03.21	22	Толстоноженко
24		3.2.3 Тестування сервера	3 дней	Вт 16.03.21	Чт 18.03.21	23	Толстоноженко
25		**3.3 Створення файлового сервера**	**25 дней**	**Пт 19.03.21**	**Чт 22.04.21**		Толстоноженко
26		3.3.1 Моделювання структури сервера	2 дней	Пт 19.03.21	Пн 22.03.21	24	Толстоноженко
27		3.3.2 Розробка сервера	20 дней	Вт 23.03.21	Пн 19.04.21	26	Толстоноженко
28		3.3.3 Тестування сервера	3 дней	Вт 20.04.21	Чт 22.04.21	27	Толстоноженко
29		3.4 Розробка взаємодії між серверами	10 дней	Пт 23.04.21	Чт 06.05.21	28	Толстоноженко
30		3.5 Загальне тестування	4 дней	Пт 07.05.21	Ср 12.05.21	29	Толстоноженко
31		**4. Завершення проекту**	**35 дней**	**Чт 13.05.21**	**Ср 30.06.21**		Толстоноженко;Шенкрик
32		4.1 Перенос проекта на хостинг	2 дней	Чт 13.05.21	Пт 14.05.21	30	Толстоноженко
33		4.2 Тестування на хостингу	2 дней	Пн 17.05.21	Вт 18.05.21	32	Толстоноженко
34		4.3 Розробка документації	4 дней	Ср 19.05.21	Пн 24.05.21	33	Толстоноженко;Шенкрик
35		4.4 Відкривання загального доступу	1 день	Вт 25.05.21	Вт 25.05.21	34	Толстоноженко
36		4.5 Перевірка на плагіат	7 дней	Ср 26.05.21	Чт 03.06.21	35	Шенкрик
37		4.6 Рецензування	7 дней	Пт 04.06.21	Пн 14.06.21	36	Толстоноженко;Шенкрик
38		4.7 Перевірка керівником	5 дней	Вт 15.06.21	Пн 21.06.21	37	Шенкрик
39		4.8 Усунення недоліків	5 дней	Чт 24.06.21	Ср 30.06.21	38	Толстоноженко

Рисунок 2.7 - Діаграма Ганта проєкту

1.6 Управління ризиками проекту

В ході реалізації проекту є ймовірність виникнення несприятливих умов, ситуацій та наслідків, які називають ризиками. Управління ризиком - це процес реагування на події та зміни ризиків у процесі виконання проекту з проведенням моніторингу ризиків. Під моніторингом ризиків розуміють процес контролю за ризиками під час створення проекту.

В процесі врахування ймовірності появи ризикових подій, що призводить до негативних наслідків роботи, створюється таблиця класифікація ризиків. Після цього створюється матриця ризиків, взявши за основу створену таблицю.

Класифікація ризиків проекту наведено на таблиці 2.2 а матриця ризиків наведено рисунок 2.8.

Таблиця 2.2 - Класифікація ризиків

№	Назва ризику	Ймовірність	Величина втрат
1	Невірно побудовано ТЗ	2	4
2	Недотримання графіку дедлайнів	4	3
3	Збої в роботі web-сервера	1	2
4	Збої в обладнання	1	3
5	Неповне тестування	3	4
6	Втрата або пошкодження даних	2	5

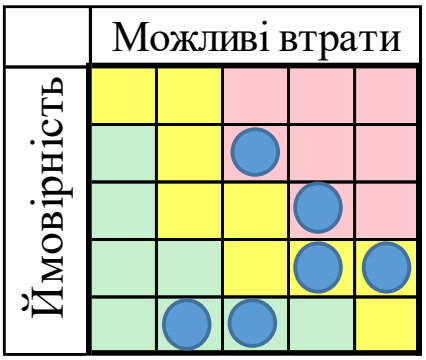


Рисунок 2.8 - Матриця ризиків

На основі дослідження матриці ризиків будуються плани реагування на ризики. Приклад такого плану наведено в таблиці 2.3.

Таблиця 2.3 – План регулювання ризиків

Ризики проекту	Реакція на ризик
Невірно побудовано ТЗ	Повторне дослідження ідеї проекту та методів її реалізації.
Недотримання графіку дедлайнів	Зміна календарного плану та розбивка на більш детальніші підзадачі.
Збої в роботі web-сервера	Зв’язок з технічною підтримкою або зміна провайдера.
Збої в обладнання	Проведення ремонту або заміни обладнання, використання системи керування версіями файлів для відновлення проекту.
Не повне тестування	Зосередити увагу на більш детальному тестуванні важливих модулів проекту.
Втрата або пошкодження даних	Регулярне створення архівів даних для відновлення в роботі над проектом при збоях або використання RAID.

ДОДАТОК В

Файли коду реалізації проекта AngularClient

Файл коду auth.interceptor.ts

```
import { HTTP_INTERCEPTORS, HttpEvent } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpHandler, HttpRequest } from '@angular/common/http';

import { TokenStorageService } from '../_services/token-storage.service';
import { Observable } from 'rxjs';

const TOKEN_HEADER_KEY = 'x-access-token';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  constructor(private token: TokenStorageService) { }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    let authReq = req;
    const token = this.token.getToken();
    if (token != null) {
      authReq = req.clone({ headers: req.headers.set(TOKEN_HEADER_KEY, token) });
    }
    return next.handle(authReq);
  }
}

export const authInterceptorProviders = [
  { provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true }
];
```

jwt.interceptor.ts

```
import {
  HttpEvent,
  HttpInterceptor,
  HttpHandler,
  HttpRequest,
  HttpErrorResponse
} from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Router } from '@angular/router';
import { Observable, throwError } from 'rxjs';
import { retry, catchError } from 'rxjs/operators';
import { TokenStorageService } from '../_services/token-storage.service';

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

  constructor(private token: TokenStorageService, private router: Router) { }
```

```
  intercept(request:         HttpRequest<any>,         next:         HttpHandler):
Observable<HttpEvent<any>> {
    return next.handle(request)
      .pipe(
        catchError((error: HttpErrorResponse) => {
          let errorMessage = '';
          if (error.error instanceof ErrorEvent) {
            errorMessage = `Error: ${error.error.message}`;
          } else {
            if ((error.status == 400 && error.error.message == "User not exist") ||
(error.status == 403 && error.error.message == "No token provided!") || error.status
== 401) {
              this.token.signOut();
              this.router.navigate(['/login']).then(() => {
                window.location.reload();
              });
              errorMessage = `Message: ${error.error}`;
            } else if(error.status == 403) {
              this.router.navigate(['/']).then(() => {
                window.location.reload();
              });
              errorMessage = `Message: ${error.error}`;
            }else{
              errorMessage     =     `Error     Code:     ${error.status}\nMessage:
${error.error}`;
            }
          }
          window.alert(errorMessage);
          return throwError(error);
        })
      )
  }
}
```

auth.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Observable } from 'rxjs';

const AUTH_API = 'http://localhost:8000/api/auth/';

const httpOptions = {
  headers: new HttpHeaders({ 'Content-Type': 'application/json' })
};

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

  constructor(private http: HttpClient) { }

  login(credentials :any): Observable<any> {
    return this.http.post(AUTH_API + 'signin', {
      email: credentials.email,
      password: credentials.password
    }, httpOptions);
  }
```

```
  register(user :any): Observable<any> {
    return this.http.post(AUTH_API + 'signup', {
      username: user.username,
      email: user.email,
      password: user.password,
      roles: ["user"]
    }, httpOptions);
  }
}
```

token-storage.service.ts

```
import { Injectable } from '@angular/core';

const TOKEN_KEY = 'auth-token';
const USER_KEY = 'auth-user';

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

  constructor() { }

  signOut(): void {
    window.sessionStorage.clear();
  }

  public saveToken(token: string): void {
    window.sessionStorage.removeItem(TOKEN_KEY);
    window.sessionStorage.setItem(TOKEN_KEY, token);
  }

  public getToken(): any {
    return  window.sessionStorage.getItem(TOKEN_KEY);
  }

  public saveUser(user: any): void {
    window.sessionStorage.removeItem(USER_KEY);
    window.sessionStorage.setItem(USER_KEY, JSON.stringify(user));
  }

  public getUser(): any {
    return  window.sessionStorage.getItem(USER_KEY);
  }
}
```

user.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Password } from '../model/Password';
import { Message } from '../model/UserMessage';
import { BlockedUser } from '../model/BlockedUser';

const API_URL = 'http://localhost:8000/api/';
const API_URL_USER = 'http://localhost:8000/api/user/';
```

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

  constructor(private http: HttpClient) { }

  getAllPaswords(): Observable<Password[]> {
    return    this.http.post<Password[]>(API_URL_USER    +    'getpasswords',    {
responseType: 'json' });
  }

  setNewPassword(password: Password): Observable<string> {
    return this.http.post(API_URL_USER + 'savepassword', {
      name: password.namePassword,
      password: password.newPassword
    }, { responseType: 'text' });
  }

  getPassword(password: Password): Observable<string> {
    return this.http.post(API_URL_USER + 'getpassword', {
      passwordId: password.id,
      password: password.oldPassword
    }, { responseType: 'text' });
  }

  resetPassword(password: Password, isResetPassword: boolean): Observable<string> {
    var body = {}
    if (isResetPassword) {
      body = {
        passwordId: password.id,
        passwordName: password.namePassword,
        password: password.oldPassword,
        passwordNew: password.newPassword
      };
    } else {
      body = {
        passwordId: password.id,
        passwordName: password.namePassword,
      };
    }
    return this.http.post(API_URL_USER + 'resetpassword',
      body,
      { responseType: 'text' });
  }

  getUsersMessage(): Observable<Message[]> {
    return   this.http.post<Message[]>(API_URL   +   'admin/getUsersMessages',   {
responseType: 'json' });
  }

  deleteUsersMessage(messageId: string): Observable<string> {
    return this.http.post(API_URL + 'admin/deleteUsersMessages', {
      messageId: messageId
    }, { responseType: 'text' });
  }

  sendMessage(message: Message): Observable<string> {
    return this.http.post(API_URL_USER + 'sendmessage', {
      name: message.name,
```

```
      description: message.description
    }, { responseType: 'text' });
  }

  getAllBlokedUsers(): Observable<BlockedUser[]> {
    return this.http.post<BlockedUser[]>(API_URL + 'admin/getAllBlockedUsers', {
responseType: 'json' });
  }

  getUserNameById(userid: string): Observable<string> {
    return this.http.post(API_URL + 'admin/getUserNameById', {
      userId: userid
    }, { responseType: 'text' });
  }

  cancelBlock(userid: string): Observable<string> {
    return this.http.post(API_URL + 'admin/cancelBlock', {
      userId: userid
    }, { responseType: 'text' });
  }

  getAllSystemMessages(): Observable<any> {
    return this.http.post(API_URL + 'admin/getAllNamesSystemMessages', {
    }, { responseType: 'json' });
  }

  getDetailsSystemMessages(fileName: string, type): Observable<any> {
    return this.http.post(API_URL + 'admin/getDetailsSystemMessages', {
      fileName: fileName
    }, { responseType: type });
  }

  deleteFile(fileName: string): Observable<string> {
    return this.http.post(API_URL + 'admin/deleteSystemFileMessage', {
      fileName: fileName
    }, { responseType: 'text' });
  }
}
```

blocked.component.html

```
<div class="container">
    <header class="jumbotron">
        {{content}}
    </header>
    <div *ngIf="blokedUsers.length != 0">
        <table class="table table-hover">
            <thead>
                <tr>
                    <th>Ім'я користувача</th>
                    <th>Дії</th>
                </tr>
            </thead>
            <tbody>
                <tr *ngFor="let user of blokedUsers">
                    <td>{{user.userName}}</td>
                    <td>
                        <button      class="btn      btn-primary      margin-right"
(click)="GetDescription(user.id)">Причина блокування</button>
```

```
                        <button              class="btn              btn-primary"
(click)="CancelBlock(user.userId)"> Розблокувати</button>
                    </td>
                </tr>
            </tbody>
        </table>
    </div>
</div>
```

blocked.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { BlockedUser } from '../model/BlockedUser';
import { TokenStorageService } from '../_services/token-storage.service';
import { UserService } from '../_services/user.service';

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

  content: string = '';
  blokedUsers: BlockedUser[] =[];
  blokedUser: BlockedUser = new BlockedUser('','','','')
  constructor(private  userService:  UserService,  private  tokenStorageService:
TokenStorageService, private router: Router) { }

  ngOnInit(): void {
    const user = JSON.parse(this.tokenStorageService.getUser());
    let roles = user?.roles;
    if(!roles || !roles.includes('ROLE_ADMIN')){
      this.router.navigate(['/']).then(() => {
        window.location.reload();
      });
    }else{
      this.GetBlokedUsers();
    }
  }

  GetBlokedUsers(): void{
    this.userService.getAllBlokedUsers().subscribe(
      data => {
        if (data.length == 0) {
          this.content = "You dont have bloked Users";
          this.blokedUsers = [];
        } else {
          this.blokedUsers = data;
        }
      },
      err => {
        JSON.parse(err.error).message;
      });
  }

  CancelBlock(idUser: string): void{
    this.userService.cancelBlock(idUser).subscribe(
      data => {
        window.alert(data);
        this.GetBlokedUsers();
```

```
      },
      err => {
        JSON.parse(err.error).message;
      });
  }

  GetDescription(id: string): void{
    let user = this.blokedUsers.find(x => x.id == id)!;
    this.content = user.description;
  }
}
```

board-user.component.html

```
<div class="container">
  <header class="jumbotron">
    <div *ngIf="IsCreate || IsUpdate || IsGetPassword; else elseBlock">
      <form (ngSubmit)="onSubmit()" #passwordForm="ngForm" autocomplete="off">
        <div class="form-group">
          <label for="">Ім'я</label>
          <input         type="text"         [(ngModel)]="newPassword.namePassword"
name="namePassword" class="form-control" id="name"
            ngModel namePassword #namePassword="ngModel" [readonly]="IsGetPassword"
required>
          <div *ngIf="!IsGetPassword && namePassword.invalid && (namePassword.dirty
|| namePassword.touched)"
            class="alert alert-danger">
            <div *ngIf="namePassword.errors!.required">
              Потрібно вказати ім’я.
            </div>
          </div>
        </div>
        <div *ngIf="IsUpdate" class="form-group">
          <input        [(ngModel)]="IsResetPassword"        name="isResetPassword"
type="checkbox"> Reset password
        </div>
        <div *ngIf="IsGetPassword || IsResetPassword" class="form-group">
          <label for="">Поточний пароль</label>
          <input        type="password"        [(ngModel)]="newPassword.oldPassword"
name="oldPassword" class="form-control"
            id="oldPassword" ngModel oldPassword #oldPassword="ngModel" required>
          <div     *ngIf="oldPassword.invalid     &&     (oldPassword.dirty     ||
oldPassword.touched)" class="alert alert-danger">
            <div *ngIf="oldPassword.errors!.required">
              Потрібний поточний пароль.
            </div>
          </div>
        </div>
        <div *ngIf="(IsUpdate && IsResetPassword) || IsCreate">
          <div class="form-group">
            <label for="">Новий пароль</label>
            <input         type="text"         [(ngModel)]="newPassword.newPassword"
name="newPasswords" class="form-control"
              id="newPasswords"   ngModel   newPasswords   #newPasswords="ngModel"
required>
            <div    *ngIf="newPasswords.invalid    &&    (newPasswords.dirty    ||
newPasswords.touched)" class="alert alert-danger">
              <div *ngIf="newPasswords.errors!.required">
                Потрібен новий пароль.
              </div>
```

```
          </div>
        </div>
        <div class="form-group">
          <label for="">Підтвердіть новий пароль</label>
          <input   type="password"   [(ngModel)]="newPassword.confirmNewPassword"
name="ConfirmNewPassword"
            class="form-control"       id="ConfirmNewPassword"       pattern="{{
newPasswords.value }}" ngModel newPassword
            #ConfirmNewPassword="ngModel" required>
          <div *ngIf="ConfirmNewPassword.invalid && (ConfirmNewPassword.dirty ||
ConfirmNewPassword.touched)"
            class="alert alert-danger">
            <div *ngIf="ConfirmNewPassword.errors!.required">
              Потрібно підтвердити пароль.
            </div>
            <div *ngIf="ConfirmNewPassword.errors!.pattern">
              Новий пароль і пароль не збігаються.
            </div>
          </div>
        </div>
      </div>
      <div *ngIf="uniqPassword" class="form-group">
        <label for="">Унікальний пароль</label>
        <div class="input-group">
          <input     type="text"     id="uniqPassword"     class="form-control"
value="{{uniqPassword}}" #formInputText readonly>
          <div class="input-group-append">
            <span class="input-group-text" id="basic-addon2">
              <ng-container *ngIf="!isCopied; else elseTemplateCopied">
                <fa-icon                          (cbOnSuccess)="copied($event)"
[ngxClipboard]="formInputText" [icon]="faCopyIcon"></fa-icon>
              </ng-container>
              <ng-template #elseTemplateCopied>
                Скопійовано
              </ng-template>
            </span>
          </div>
        </div>
      </div>
      <div class="form-group">
        <button          class="btn          btn-success          margin-right"
[disabled]="!passwordForm.form.valid        ||        (!IsResetPassword        &&
!(newPassword.namePassword               !=               password.namePassword))"
type="submit">Надіслати</button>
        <button          class="btn          btn-danger          margin-right"
(click)="CanselForm()">Скасувати</button>
      </div>
    </form>
  </div>
  <ng-template #elseBlock>
    <p>{{ content }}</p>
  </ng-template>
</header>
<button class="btn btn-primary" (click)="AddPassword()"> Додати пароль</button>
<div *ngIf="passwords.length != 0">
  <table class="table table-hover">
    <thead>
      <tr>
        <th hidden>id</th>
        <th>Ім'я пароля</th>
        <th>Дії</th>
```

```
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let passwor of passwords">
          <td hidden> passwor.id</td>
          <td> {{ passwor.namePassword }}</td>
          <td>
            <button          class="btn          btn-primary          margin-right"
(click)="GetPassword(passwor.id)"> Отримати</button>
            <button    class="btn    btn-primary"    (click)="Update(passwor.id)">
Змінити</button>
          </td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
```

board-user.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Password } from '../model/Password';
import { UserService } from '../_services/user.service';
import { FormBuilder } from '@angular/forms';
import { faCopy } from '@fortawesome/free-solid-svg-icons';

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

  faCopyIcon = faCopy;
  content: string = '';
  passwords: Password[] = [];
  IsCreate: boolean = false;
  IsGetPassword: boolean = false;
  IsUpdate: boolean = false;
  IsResetPassword: boolean = false;
  password: Password = new Password('', '','','','');
  uniqPassword: string = '';
  isCopied = false;
  newPassword: Password = new Password('', '','','','');

  constructor(private formBuilder: FormBuilder, private userService: UserService) {

  }

  ngOnInit(): void {
    this.GetAllPasswords();
  }

  GetAllPasswords(): void {
    this.userService.getAllPaswords().subscribe(
      data => {
        if (data.length == 0) {
          this.content = "У вас немає паролів";
        } else {
          this.passwords = data;
```

```
      }
    },
    err => {
      this.content = JSON.parse(err.error).message;
    }
  );
}

GetPassword(id: string): void {
  this.CanselForm();
  this.IsGetPassword = !this.IsGetPassword;
  this.newPassword = Object.assign({}, this.passwords.find(x => x.id == id)!);
}

copied(event) {
  this.isCopied = event.isSuccess;
}

Update(id: string): void {
  this.CanselForm();
  this.IsUpdate = !this.IsUpdate;
  this.password = this.passwords.find(x => x.id == id)!;
  this.newPassword = Object.assign({}, this.password);

}

AddPassword(): void {
  this.CanselForm();
  this.IsCreate = !this.IsCreate;
}

CanselForm(): void {
  this.IsUpdate = false;
  this.IsCreate = false;
  this.isCopied = false;
  this.IsGetPassword = false;
  this.IsResetPassword = false;
  this.password = new Password('', '','','','');
  this.newPassword = new Password('', '','','','');
  this.uniqPassword = '';
  this.content = '';
}

onSubmit() {
  if (this.IsCreate) {
    this.userService.setNewPassword(this.newPassword).subscribe(
      result => {
        this.GetAllPasswords();
        this.uniqPassword = result;

      }, err => {
        this.content = JSON.parse(err.error).message;
      }
    );
  }else if(this.IsGetPassword){
    this.userService.getPassword(this.newPassword).subscribe(
      result => {
        this.uniqPassword = result;
      }, err => {
        this.content = JSON.parse(err.error).message;
      }
```

```
      );
    }else if(this.IsUpdate){
      this.userService.resetPassword(this.newPassword,
this.IsResetPassword).subscribe(
        result => {
          if(result == "changed"){
            this.content  = 'Changed';
          }else{
            this.uniqPassword = result;
          }
          this.GetAllPasswords();
        }, err => {
          this.content = JSON.parse(err.error).message;
        }
      );
    }
  }
}
```

bug-report-system.component.html

```
<div class="container">
    <header class="jumbotron">
      <pre>{{ content | json }}</pre>
    </header>
  </div>

  <table class="table table-hover">
    <thead>
      <tr>
        <th>Назва</th>
        <th>Дії</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let message of messages">
        <td> {{ message }}</td>
        <td>
          <button        class="btn        btn-primary        button-margin-right"
(click)="selectMessage(message)"> Показати деталі</button>
          <button        class="btn        btn-success        button-margin-right"
(click)="DownloadFile(message)"> Завантажити</button>
          <button     class="btn     btn-danger"     (click)="DeleteFile(message)">
Видалити</button>
        </td>
      </tr>
    </tbody>
  </table>
```

bug-report-system.component.ts

```
import { Component, OnInit } from '@angular/core';
import { UserService } from '../_services/user.service';
import * as fileSaver from 'file-saver';
import { TokenStorageService } from '../_services/token-storage.service';
import { Router } from '@angular/router';

@Component({
```

```
  selector: 'app-bug-report',
  templateUrl: './bug-report-system.component.html',
  styleUrls: ['./bug-report-system.component.css']
})
export class BugReportSystemComponent implements OnInit {

  content: any = 'Виберіть повідомлення';
  messages: string[] = [];

  constructor(private userService: UserService , private tokenStorageService:
TokenStorageService, private router: Router) { }

  ngOnInit(): void {
    const user = JSON.parse(this.tokenStorageService.getUser());
    let roles = user?.roles;
    if(!roles || !roles.includes('ROLE_ADMIN')){
      this.router.navigate(['/']).then(() => {
        window.location.reload();
      });
    }else{
      this.getAllSystemMessages();
    }
  }

  getAllSystemMessages(){
    this.userService.getAllSystemMessages().subscribe(
      data => {
        this.messages = data;
      },
      err => {
        this.content = JSON.parse(err.error).message;
      }
    );
  }

  selectMessage(fileName: string): void {
    this.userService.getDetailsSystemMessages(fileName, 'json').subscribe(
      data => {
        this.content = data;
      },
      err => {
        this.content = JSON.parse(err.error).message;
      }
    );
  }

  DownloadFile(fileName: string): void {
    this.userService.getDetailsSystemMessages(fileName, 'blob').subscribe(
      data => {
        fileSaver.saveAs(data, fileName);
      },
      err => {
        this.content = JSON.parse(err.error).message;
      }
    );
  }

  DeleteFile(fileName: string): void {
    this.userService.deleteFile(fileName).subscribe(
      data => {
        this.content = data;
```

```
        this.getAllSystemMessages();
      },
      err => {
        this.content = JSON.parse(err.error).message;
      }
    );
  }
}
```

bug-report-users.component.html

```
<div class="container">
  <header class="jumbotron">
    <p>{{ content }}</p>
  </header>
</div>

<table class="table table-hover">
  <thead>
    <tr>
      <th>Назва</th>
      <th>Ім'я користувача</th>
      <th>Дії</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let messag of messages">
      <td> {{ messag.name }}</td>
      <td> {{ messag.userName }}</td>
      <td>
        <button         class="btn         btn-primary         button-margin-right"
(click)="selectMessage(messag.id)"> Показати деталі</button>
        <button    class="btn    btn-danger"    (click)="deleteMessage(messag.id)">
Видалити</button>
      </td>
    </tr>
  </tbody>
</table>
```

bug-report-users.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { Message } from '../model/UserMessage';
import { TokenStorageService } from '../_services/token-storage.service';
import { UserService } from '../_services/user.service';

@Component({
  selector: 'bug-report-users',
  templateUrl: './bug-report-users.component.html',
  styleUrls: ['./bug-report-users.component.css']
})
export class BugReportUsersComponent implements OnInit {

  content: string ='Виберіть повідомлення';
  messages: Message[] = [];
  message: Message = new Message('','','','');
```

```
  constructor(private userService: UserService, private tokenStorageService:
TokenStorageService, private router: Router) { }

  ngOnInit(): void {
    const user = JSON.parse(this.tokenStorageService.getUser());
    let roles = user?.roles;
    if(!roles || !roles.includes('ROLE_ADMIN')){
      this.router.navigate(['/']).then(() => {
        window.location.reload();
      });
    }else{
      this.getUsersMessage();
    }
  }

  getUsersMessage(){
    this.userService.getUsersMessage().subscribe(
      data => {
        this.messages = data;
      },
      err => {
        this.content = JSON.parse(err.error).message;
      }
    );
  }

  selectMessage(id:string): void{
    this.message = this.messages.find(x => x.id == id)!;
    this.content = this.message.description;
  }

  deleteMessage(id:string): void{
    this.userService.deleteUsersMessage(id).subscribe(
      data => {
        this.content = data;
        this.getUsersMessage();
      },
      err => {
        this.content = JSON.parse(err.error).message;
      }
    );
  }

}
```

home.component.html

```
<div class="container">
  <div class="text-center"><h3>Привіт, це "Safe Password Storage".<br>Ви можете
протестувати проект.</h3></div>
  <header class="jumbotron">
    <form (ngSubmit)="onSubmit()" #passwordForm="ngForm" autocomplete="off">
        <div class="form-group">
          <label for="">Пароль</label>
          <input type="text" [(ngModel)]="password.newPassword" name="newPasswords"
class="form-control"
            id="newPasswords"    ngModel    newPasswords    #newPasswords="ngModel"
required>
          <div     *ngIf="newPasswords.invalid     &&     (newPasswords.dirty     ||
newPasswords.touched)" class="alert alert-danger">
```

```
            <div *ngIf="newPasswords.errors!.required">
              Потрібен пароль.
            </div>
          </div>
        </div>
        <div *ngIf="uniqPassword" class="form-group">
          <label for="">Унікальний пароль</label>
          <div class="input-group">
            <input      type="text"      id="uniqPassword"      class="form-control"
value="{{uniqPassword}}" #formInputText readonly>
          </div>
        </div>
        <div class="form-group">
          <button class="btn btn-success margin-right"
            [disabled]="!passwordForm.form.valid || uniqPassword"
            type="submit">Отримати</button>
        </div>
    </form>
  </header>
</div>
```

home.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Password } from '../model/Password';
import { UserService } from '../_services/user.service';

declare function Encrypt(params:string):any;

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

  content: string = '';
  password: Password = new Password('', '','','','');
  uniqPassword: string = '';

  constructor(private userService: UserService) { }

  ngOnInit(): void {
  }

  onSubmit() {
    this.uniqPassword = Encrypt(this.password.newPassword);
  }
}
```

login.component.html

```
<aside class="col-md-5 mx-auto vertical-center">
  <div class="card">
    <article class="card-body">
      <form *ngIf="!isLoggedIn" name="form" (ngSubmit)="f.form.valid && onSubmit()"
#f="ngForm" novalidate>
        <div class="form-group">
```

```
          <label for="email">Електронна пошта</label>
          <input        type="email"        class="form-control"        name="email"
[(ngModel)]="form.email" required #email="ngModel" />
          <div class="alert-danger" *ngIf="f.submitted && email.invalid">
            <div *ngIf="email.errors?.required">Потрібна електронна адреса</div>
            <div *ngIf="email.errors?.email">
              Електронна адреса повинна відповідати шаблону електронної адреси
            </div>
          </div>
        </div>
        <div class="form-group">
          <label for="password">Пароль</label>
          <input      type="password"      class="form-control"      name="password"
[(ngModel)]="form.password" required minlength="6"
            #password="ngModel" />
          <div  class="alert  alert-danger"  role="alert"  *ngIf="f.submitted  &&
password.invalid">
            <div *ngIf="password.errors?.required">Потрібен пароль</div>
            <div *ngIf="password.errors?.minlength">
              Пароль повинен містити щонайменше 6 символів
            </div>
          </div>
        </div>
        <div class="form-group">
          <button class="btn btn-primary btn-block">
            Увійти
          </button>
        </div>
        <div class="form-group">
          <div  class="alert  alert-danger"  role="alert"  *ngIf="f.submitted  &&
isLoginFailed">
            Помилка логіну:{{ errorMessage }}
          </div>
        </div>
      </form>

      <div class="alert alert-success" *ngIf="isLoggedIn">
        Ви увійшли як {{ roles }}.
      </div>
    </article>
  </div>
</aside>
```

login.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from '../_services/auth.service';
import { TokenStorageService } from '../_services/token-storage.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {
  form: any = {};
  isLoggedIn = false;
  isLoginFailed = false;
  errorMessage = '';
```

```
  roles: string[] = [];

  constructor(  private  authService:  AuthService,  private  tokenStorage:
TokenStorageService, private router: Router) {}

  ngOnInit(): void {
    if (this.tokenStorage.getToken()) {
      this.isLoggedIn = true;
      this.roles = this.tokenStorage.getUser().roles;
    }
  }

  onSubmit(): void {
    this.authService.login(this.form).subscribe(
      data => {
        this.tokenStorage.saveToken(data.accessToken);
        this.tokenStorage.saveUser(data);

        this.isLoginFailed = false;
        this.isLoggedIn = true;
        this.roles = this.tokenStorage.getUser().roles;
        this.router.navigate(['/']).then(() => {
          window.location.reload();
        });
      },
      err => {
        this.errorMessage = err.error.message;
        this.isLoginFailed = true;
      }
    );
  }

  reloadPage(): void {
    window.location.reload();
  }

}
```

BlockedUser.ts

```
export class BlockedUser{
    id:string;
    userId:string;
    userName:string;
    description: string;

    constructor(id:string, userId: string, userName: string, description: string) {
        this.id = id;
        this.userName = userName;
        this.userId = userId;
        this.description = description;
    }
}
```

Password.ts

```
export class Password{
    id:string;
```

```
    namePassword:string;
    newPassword: string;
    oldPassword: string;
    confirmNewPassword: string;

    constructor(id:string, namePassword: string, newPassword: string, oldPassword:
string, confirmNewPassword: string) {
        this.id = id;
        this.namePassword = namePassword;
        this.newPassword = newPassword;
        this.oldPassword = oldPassword;
        this.confirmNewPassword = confirmNewPassword;
    }
}
```

SystemMessage.ts

```
export class SystemMessage{
    id:string;
    name:string;

    constructor(id:string, name: string) {
        this.id = id;
        this.name = name;
    }
}
```

UserMessage.ts

```
export class Message{
    id:string;
    name:string;
    userName: string;
    description: string;

    constructor(id:string, name: string, userName: string, description: string) {
        this.id = id;
        this.name = name;
        this.userName = userName;
        this.description = description;
    }
}
```

profile.component.html

```
<div class="container" *ngIf="currentUser; else loggedOut">
    <header class="jumbotron">
      <h3>
        <strong>{{ currentUser.username }}</strong> Профіль
      </h3>
    </header>
    <p>
      <strong>Token:</strong>
      {{ currentUser.accessToken.substring(0, 20) }} ...
      {{ currentUser.accessToken.substr(currentUser.accessToken.length - 20) }}
    </p>
    <p>
```

```
      <strong>Email:</strong>
      {{ currentUser.email }}
    </p>
    <strong>Roles:</strong>
    <ul>
      <li *ngFor="let role of currentUser.roles">
        {{ role }}
      </li>
    </ul>
  </div>

  <ng-template #loggedOut>
    Please login.
  </ng-template>
```

profile.component.ts

```
import { Component, OnInit } from '@angular/core';
import { TokenStorageService } from '../_services/token-storage.service';

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

  currentUser: any;

  constructor(private token: TokenStorageService) { }

  ngOnInit(): void {
    this.currentUser = JSON.parse(this.token.getUser());
  }
}
```

register.component.html

```
<aside class="col-md-5 mx-auto vertical-center">
  <div class="card">
    <article class="card-body">
      <form   *ngIf="!isSuccessful"   name="form"   (ngSubmit)="f.form.valid   &&
onSubmit()" #f="ngForm" novalidate>
        <div class="form-group">
          <label for="username">Ім'я користувача</label>
          <input       type="text"       class="form-control"       name="username"
[(ngModel)]="form.username" required minlength="3"
            maxlength="20" #username="ngModel" />
          <div class="alert-danger" *ngIf="f.submitted && username.invalid">
            <div *ngIf="username.errors?.required">Потрібно ім'я користувача</div>
            <div *ngIf="username.errors?.minlength">
              Ім'я користувача має містити принаймні 3 символи
            </div>
            <div *ngIf="username.errors?.maxlength">
              Ім'я користувача має містити не більше 20 символів
            </div>
          </div>
        </div>
```

```
        <div class="form-group">
          <label for="email">Електронна пошта</label>
          <input        type="email"        class="form-control"        name="email"
[(ngModel)]="form.email" required email
            #email="ngModel" />
          <div class="alert-danger" *ngIf="f.submitted && email.invalid">
            <div *ngIf="email.errors?.required">Потрібна електронна пошта</div>
            <div *ngIf="email.errors?.email">
              Електронна адреса повинна відповідати шаблону електронної адреси
            </div>
          </div>
        </div>
        <div class="form-group">
          <label for="password">Пароль</label>
          <input       type="text"       class="form-control"       name="password"
[(ngModel)]="form.password" required minlength="6"
            #password="ngModel" />
          <div class="alert-danger" *ngIf="f.submitted && password.invalid">
            <div *ngIf="password.errors?.required">Потрібен пароль</div>
            <div *ngIf="password.errors?.minlength">
              Пароль повинен містити щонайменше 6 символів
            </div>
          </div>
        </div>
        <div class="form-group">
          <button class="btn btn-primary btn-block">Зареєструватися</button>
        </div>

        <div class="alert alert-warning" *ngIf="f.submitted && isSignUpFailed">
          Помилка реєстрації!<br />{{ errorMessage }}
        </div>
      </form>

      <div class="alert alert-success" *ngIf="isSuccessful">
        Ваша реєстрація успішна!
      </div>
    </article>
  </div>
</aside>
```

register.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from '../_services/auth.service';

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

  form: any = {};
  isSuccessful = false;
  isSignUpFailed = false;
  errorMessage = '';
```

```
  constructor(private authService: AuthService, private router: Router) { }

  ngOnInit(): void {
  }

  onSubmit(): void {
    this.authService.register(this.form).subscribe(
      data => {
        this.isSuccessful = true;
        this.isSignUpFailed = false;
        this.router.navigate(['/login']).then(() => {
          window.location.reload();
        });
      },
      err => {
        this.errorMessage = err.error.message;
        this.isSignUpFailed = true;
      }
    );
  }

}
```

report.component.html

```
<form (ngSubmit)="onSubmit()" #messageForm="ngForm" autocomplete="off">
    <div class="form-group">
        <label for="">Ім'я</label>
        <input     type="text"     [(ngModel)]="message.name"     name="nameMessage"
class="form-control" id="nameMessage" ngModel
            nameMessage #nameMessage="ngModel" required>
        <div      *ngIf="nameMessage.invalid      &&      (nameMessage.dirty      ||
nameMessage.touched)" class="alert alert-danger">
            <div *ngIf="nameMessage.errors!.required">
                Потрібно вказати ім’я.
            </div>
        </div>
    </div>
    <div class="form-group">
        <label for="">Опис</label>
        <textarea [(ngModel)]="message.description" name="description" class="form-
control" id="description" ngModel
        description #description="ngModel" required></textarea>
        <div      *ngIf="description.invalid      &&      (description.dirty      ||
description.touched)" class="alert alert-danger">
            <div *ngIf="description.errors!.required">
                Опис обов’язковий.
            </div>
        </div>
    </div>
    <div class="form-group">
        <button           class="btn           btn-success           margin-right"
[disabled]="!messageForm.form.valid" type="submit">Надіслати</button>
        <button class="btn btn-danger" (click)="CanselForm()">Скасувати</button>
      </div>
</form>
```

report.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { Message } from '../model/UserMessage';
import { TokenStorageService } from '../_services/token-storage.service';
import { UserService } from '../_services/user.service';

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

  message:Message = new Message('','','','');

  constructor(private tokenStorageService: TokenStorageService, private router:
Router, private userService: UserService) { }

  ngOnInit(): void {
    const user = JSON.parse(this.tokenStorageService.getUser());
    let roles = user.roles;
    if(!roles.includes('ROLE_USER')){
      this.router.navigate(['/']).then(() => {
        window.location.reload();
      });
    }
  }
  onSubmit() {
    this.userService.sendMessage(this.message).subscribe(
      data => {
        window.alert(data);
        this.CanselForm();
      });
  }
  CanselForm(){
    this.router.navigate(['/home']).then(() => {
      window.location.reload();
    });
  }
}
```

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { RegisterComponent } from './register/register.component';
import { LoginComponent } from './login/login.component';
import { HomeComponent } from './home/home.component';
import { ProfileComponent } from './profile/profile.component';
import { BoardUserComponent } from './board-user/board-user.component';
import { BugReportUsersComponent } from './bug-report-users/bug-report-
users.component';
import { ReportComponent } from './report/report.component';
import { BlockedComponent } from './blocked/blocked.component';
import { BugReportSystemComponent } from './bug-report-system/bug-report-
system.component';
```

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'login', component: LoginComponent },
  { path: 'register', component: RegisterComponent },
  { path: 'profile', component: ProfileComponent },
  { path: 'user', component: BoardUserComponent },
  { path: 'bug-report-users', component: BugReportUsersComponent },
  { path: 'report', component: ReportComponent },
  { path: 'blocked', component: BlockedComponent },
  { path: 'bug-report-system', component: BugReportSystemComponent },
  { path: '', redirectTo: 'home', pathMatch: 'full' }
];

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

app.component.html

```
<div class="app">
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <a href="#" class="navbar-brand">Safe Password Storage</a>
    <button class="navbar-toggler btn btn-outline-primary" type="button" data-toggle="collapse" data-target="#navbarTogglerDemo02" aria-expanded="false" aria-label="Toggle navigation" (click)="isCollapsed = !isCollapsed" [attr.aria-expanded]="!isCollapsed" aria-controls="navbarTogglerDemo02">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarTogglerDemo02" [ngbCollapse]="isCollapsed">
      <ul class="navbar-nav mr-auto" routerLinkActive="active" (click)="isCollapsed = !isCollapsed">
        <li class="nav-item">
          <a href="/home" class="nav-link" routerLink="home">Додому </a>
        </li>
        <li class="nav-item" *ngIf="showAdminBoard">
          <a href="/admin" class="nav-link" *ngIf="isLoggedIn" routerLink="bug-report-users">Звіти про помилки (користувачі)</a>
        </li>
        <li class="nav-item" *ngIf="showAdminBoard">
          <a href="/admin" class="nav-link" *ngIf="isLoggedIn" routerLink="bug-report-system">Звіти про помилки (система)</a>
        </li>
        <li class="nav-item" *ngIf="showAdminBoard">
          <a href="/blocked" class="nav-link" *ngIf="isLoggedIn" routerLink="blocked">Заблоковані користувачі</a>
        </li>
        <li class="nav-item" *ngIf="showUserBoard">
          <a href="/user" class="nav-link" *ngIf="isLoggedIn" routerLink="user">Паролі</a>
        </li>
        <li class="nav-item" *ngIf="showUserBoard">
          <a href="/report" class="nav-link" *ngIf="isLoggedIn" routerLink="report">Повідомити про помилку</a>
        </li>
      </ul>
```

```
      <ul class="navbar-nav ml-auto" *ngIf="!isLoggedIn" (click)="isCollapsed =
!isCollapsed">
        <li class="nav-item">
          <a                    href="/register"                    class="nav-link"
routerLink="register">Зареєструватися</a>
        </li>
        <li class="nav-item">
          <a href="/login" class="nav-link" routerLink="login">Увійти</a>
        </li>
      </ul>

      <ul class="navbar-nav ml-auto" *ngIf="isLoggedIn" (click)="isCollapsed =
!isCollapsed">
        <li class="nav-item">
          <a href="/profile" class="nav-link" routerLink="profile">{{ username
}}</a>
        </li>
        <li class="nav-item">
          <a href class="nav-link" (click)="logout()">Вийти</a>
        </li>
      </ul>
    </div>
  </nav>
  <div class="container">
    <router-outlet></router-outlet>
  </div>
</div>
```

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { TokenStorageService } from './_services/token-storage.service';
import { DeviceDetectorService } from 'ngx-device-detector';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  aboutProjectOnPC = {
    nameProject: "Safe Password Storage",
    report: "Bug reports",
    blocked: "Blocked users"
  };
  aboutProjectOnMob = {
    nameProject: "SPS",
    report: "B_reports",
    blocked: "B_users"
  };
  isCollapsed = true;
  aboutProject = this.aboutProjectOnPC;
  private roles: string[] = [];
  isLoggedIn = false;
  showAdminBoard = false;
  showUserBoard = false;
  username: string = '';

  constructor(private     tokenStorageService:     TokenStorageService,     private
deviceService: DeviceDetectorService) { }
```

```
  ngOnInit(): void {
    this.isLoggedIn = !!this.tokenStorageService.getToken();
    if (this.isLoggedIn) {
      const user = JSON.parse(this.tokenStorageService.getUser());
      this.roles = user.roles;

      this.showAdminBoard = this.roles?.includes('ROLE_ADMIN');
      this.showUserBoard = this.roles?.includes('ROLE_USER');

      this.username = user.username;
    }
  }

  logout(): void {
    this.tokenStorageService.signOut();
    window.location.reload();
  }
}
```

app.module.ts

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

import { AppRoutingModule } from './app-routing.module';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';

import { AppComponent } from './app.component';
import { LoginComponent } from './login/login.component';
import { RegisterComponent } from './register/register.component';
import { HomeComponent } from './home/home.component';
import { ProfileComponent } from './profile/profile.component';
import { BugReportUsersComponent } from './bug-report-users/bug-report-
users.component';
import { BoardUserComponent } from './board-user/board-user.component';

import { authInterceptorProviders } from './_helpers/auth.interceptor'
import { ClipboardModule } from 'ngx-clipboard';
import { FontAwesomeModule } from '@fortawesome/angular-fontawesome';
import { HttpErrorInterceptor } from './_helpers/jwt.interceptor';
import { ReportComponent } from './report/report.component';
import { BlockedComponent } from './blocked/blocked.component';
import {NgbModule} from '@ng-bootstrap/ng-bootstrap';
import { BugReportSystemComponent } from './bug-report-system/bug-report-
system.component';

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    RegisterComponent,
    HomeComponent,
    ProfileComponent,
    BugReportUsersComponent,
    BoardUserComponent,
    ReportComponent,
    BlockedComponent,
    BugReportSystemComponent
```

```
  ],
  imports: [
    BrowserModule,
    ReactiveFormsModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule,
    ClipboardModule,
    FontAwesomeModule,
    NgbModule
  ],
  providers: [
    authInterceptorProviders,
    {
      provide: HTTP_INTERCEPTORS,
      useClass: HttpErrorInterceptor,
      multi: true
    }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Файли коду реалізації проекта ClientServer

auth.config.js

```
module.exports = {
      secret: "SrUa6sa1kds"
};
```

Client.js

```
const {Keys} =  require('../crypt/rsa');
export class Client {
    socket_id;
    client_ip;
    keys;

    constructor(socket_id, client_ip){
        this.client_ip = client_ip;
        this.socket_id = socket_id;
    }

    SetKeys(keys){
        this.keys = new Keys(keys);
    }

    CheckUser(socket_id, client_ip){
        return (this.client_ip == client_ip) && (this.socket_id == socket_id);
    }
}
```

db.config.js

```
module.exports = {
```

```
      HOST: "localhost",
      USER: "root",
      PASSWORD: "root",
      DB: "testdb",
      dialect: "mysql",
      pool: {
            max: 5,
            min: 0,
            acquire: 30000,
            idle: 10000
      }
};
```

auth.controller.js

```
const db = require("../models");
const config = require("../config/auth.config");
const User = db.user;
const Role = db.role;

const Op = db.Sequelize.Op;

var jwt = require("jsonwebtoken");
var bcrypt = require("bcryptjs");

exports.signup = (req, res) => {
    // Save User to Database
    User.create({
        username: req.body.username,
        email: req.body.email,
        password: bcrypt.hashSync(req.body.password, 8)
    })
        .then(user => {
            // user role = 1
            user.setRoles([1]).then(() => {
                res.send({ message: "Користувач був успішно зареєстрований!" });
            });
        })
        .catch(err => {
            res.status(500).send({ message: err.message });
        });
};

exports.signin = (req, res) => {
    User.findOne({
        where: {
            email: req.body.email
        }
    })
        .then(user => {
            if (!user) {
                return res.status(404).send({ message: "Користувач не знайдений."
});
            }

            var passwordIsValid = bcrypt.compareSync(
                req.body.password,
                user.password
            );
```

```
            if (!passwordIsValid) {
                return res.status(401).send({
                    accessToken: null,
                    message: "Недійсний пароль!"
                });
            }

            var token = jwt.sign({ id: user.id }, config.secret, {
                expiresIn: 900 // 15 min
            });

            var authorities = [];
            user.getRoles().then(roles => {
                for (let i = 0; i < roles.length; i++) {
                    authorities.push("ROLE_" + roles[i].name.toUpperCase());
                }
                res.status(200).send({
                    username: user.username,
                    email: user.email,
                    roles: authorities,
                    accessToken: token
                });
            });
        })
        .catch(err => {
            res.status(500).send({ message: err.message });
        });
};
```

user.controller.js

```
const db = require("../models");
const config = require("../config/auth.config");
const User = db.user;
const Password = db.password;
const Message = db.message;
const BlockedUser = db.blockedUser

exports.allAccess = (req, res) => {
      res.status(200).send("Public Content.");
};

exports.userBoard = (req, res) => {
      res.status(200).send("User Content.");
};

exports.adminBoard = (req, res) => {
      res.status(200).send("Admin Content.");
};

exports.resetPasswords = (req, res) => {
      Password.findByPk(req.body.passwordId).then(password => {
            if (!password) {
                  return res.status(404).send({ message: "Пароль не знайдено." });
            }
            if (password.userId != req.userId) {
                  return    res.status(404).send({    message:    "Ви    не    можете
використовувати його" });
            }
            password.update({
```

```
                  name: req.body.passwordName
            }).then(result => {
                  if (req.body.passwordNew) {
                        return res.send(req.unicPassword);
                  }
                  return res.send("changed");
            });
      });
};

exports.getfilesID = (req, res, next) => {
      Password.findByPk(req.body.passwordId).then(password => {
            if (!password) {
                  return res.status(404).send({ message: "Пароль не знайдено." });
            }
            if (password.userId != req.userId) {
                  return    res.status(404).send({    message:    "Ви    не    можете
використовувати його" });
            }
            req.filesId = {
                  fileId: password.file_password_id,
                  webSiteId: password.web_site_id,
            };
            next();
            return;
      });
};

exports.getPasswords = (req, res) => {
      User.findByPk(req.userId).then(user => {
            if (!user) {
                  return res.status(404).send({ message: "Користувач не знайдений."
});
            }
            user.getPasswords().then(passwords => {
                  let passwordsResult = [];
                  if (passwords.length > 0) {
                        for (let i = 0; i < passwords.length; i++) {
                              passwordsResult.push({
                                    id: passwords[i].id,
                                    namePassword: passwords[i].name
                              });
                        }
                        res.send(passwordsResult);
                  } else {
                        res.status(200).send(passwordsResult);
                  }
            });
      });
};

exports.savePassword = (req, res) => {
      User.findByPk(req.userId).then(user => {
            if (!user) {
                  return res.status(404).send({ message: "Користувач не знайдений."
});
            }
            user.createPassword({
                  name: req.body.name,
                  file_password_id: req.unicPassword.id,
                  web_site_id: req.unicPassword.webId
```

```
            }).then(result => {
                  res.send(req.unicPassword.password);
            }).catch(err => {
                  console.log(err)
            });
      });
};

exports.sendmessage = (req, res) => {
      User.findByPk(req.userId).then(user => {
            if (!user) {
                  return res.status(404).send({ message: "Користувач не знайдений."
});
            }
            user.createMessage({
                  name: req.body.name,
                  userName: user.username,
                  description: req.body.description
            }).then(result => {
                  res.send("Повідомлення надіслано");
            }).catch(err => {
                  console.log(err)
            });
      });
};

exports.getMessages = (req, res) => {
      Message.findAll().then(messages => {
            let messagesResult = [];
            if (messages.length > 0) {
                  for (let i = 0; i < messages.length; i++) {
                        messagesResult.push({
                              id: messages[i].id,
                              name: messages[i].name,
                              userName: messages[i].userName,
                              description: messages[i].description
                        });
                  }
                  res.send(messagesResult);
            } else {
                  res.status(200).send(messagesResult);
            }
      });
};

exports.deleteUsersMessages = (req, res) => {
      Message.destroy({
            where: {
                  id: req.body.messageId
            }
      }).then(function (rowDeleted) { // rowDeleted will return number of rows
deleted
            if (rowDeleted === 1) {
                  res.send('Видалено успішно');
            }
      }, function (err) {
            cres.send(err);
      });
};

exports.cancelBlock = (req, res) => {
```

```
      BlockedUser.destroy({
            where: {
                  userId: req.body.userId
            }
      }).then(result => {
            res.status(200).send("Розблоковано");
      });
};


exports.getAllBlockedUsers = (req, res) => {
      BlockedUser.findAll().then(blockedUsers => {
            let blockedUsersResult = [];
            if (blockedUsers.length > 0) {
                  for (let i = 0; i < blockedUsers.length; i++) {
                        blockedUsersResult.push({
                              id: blockedUsers[i].id,
                              userId: blockedUsers[i].userId,
                              userName: blockedUsers[i].userName,
                              description: blockedUsers[i].description
                        });
                  }
                  res.send(blockedUsersResult);
            } else {
                  res.status(200).send(blockedUsersResult);
            }
      });
};

exports.getUserNameById = (req, res) => {
      User.findByPk(req.userId).then(user => {
            res.send(user.username);
      });
};
```

rsa.js

```
var bigInt = require("big-integer");
export class Keys {
    publicKey;
    n;

    constructor(obj){
        Object.assign(this, obj)
      }

    SetKeys(publicKey, n) {
        this.publicKey = publicKey;
        this.n = n;
    }
}

export class RSA extends Keys {
    #privateKey;
    constructor(p, q) {
        super();
        this.p = p;
        this.q = q;
    }
```

```
createKey() {
    this.n = this.p * this.q;
    var u = (this.p - 1) * (this.q - 1);
    var primeNumbers = [];
    for (var i = 11; i < 5000; i++) {
        if (this.IsPrimeNumber(i)) {
            primeNumbers.push(i);
        }
    }
    const random = Math.floor(Math.random() * primeNumbers.length);
    this.publicKey = primeNumbers[random];

    this.#privateKey = 1;
    do {
        this.#privateKey++;
    } while ((this.#privateKey * this.publicKey) % u != 1);

    console.log("Private: " + this.#privateKey + " Public: " + this.publicKey);
    var key = new Keys();
    key.SetKeys(this.publicKey, this.n);
    return key;
}

IsPrimeNumber(n) {
    var result = true;
    if (n > 1) {
        for (var i = 2; i < n; i++) {
            if (n % i == 0) {
                result = false;
                break;
            }
        }
    }
    else {
        result = false;
    }

    return result;
}

Encription(text, key) {
    let arr = new Array();
    var enc = text.split('');
    for (var i = 0; i < enc.length; i++) {
        var t = enc[i].charCodeAt(0);
        var bigIn1 = Number(bigInt(t).pow(key.publicKey).mod(key.n));
        arr.push(bigIn1);
    }
    console.log("Array1: " + arr.join(','));
    return arr;
}

Dencription(arr) {
    console.log("Array2: " + arr.join(','));
    let textC = new Array(arr.length);
    for (var i = 0; i < textC.length; i++) {
        var bigIn1 = Number(bigInt(arr[i]).pow(this.#privateKey).mod( this.n));
        textC[i] = String.fromCharCode(bigIn1);
    }
    let result = textC.join('');
```

```
        return result;
    }

}
```

authJwt.js

```
const jwt = require("jsonwebtoken");
const config = require("../config/auth.config.js");
const db = require("../models");
const User = db.user;
const BlockedUser = db.blockedUser;

function verifyToken(req, res, next) {
      let token = req.headers["x-access-token"];

      if (!token) {
            return res.status(403).send({
                  message: "No token provided!"
            });
      }

      jwt.verify(token, config.secret, (err, decoded) => {
            if (err) {
                  if (err.message == 'jwt expired') {
                        return res.status(401).send("Time working token lost!");
                  }
                  return res.status(401).send("Unauthorized!");
            }
            req.userId = decoded.id;
            isUserExist(req, res, next);
      });
};

function isUserExist(req, res, next) {
      User.findByPk(req.userId).then(user => {
            if (user) {
                  next();
                  return;
            } else {
                  res.status(400).send({
                        message: "User not exist"
                  });
            }
      })
}

function isAdmin(req, res, next) {
      User.findByPk(req.userId).then(user => {
            user.getRoles().then(roles => {
                  for (let i = 0; i < roles.length; i++) {
                        if (roles[i].name === "admin") {
                              next();
                              return;
                        }
                  }

                  res.status(403).send({
                        message: "Require Admin Role!"
                  });
```

```
                return;
            });
      });
};

function isUser(req, res, next) {
      User.findByPk(req.userId).then(user => {
            user.getRoles().then(roles => {
                  for (let i = 0; i < roles.length; i++) {
                        let isUser = false;
                        if (roles[i].name === "user") {
                              isUser = true;
                              let test = user.getBlockedUser().then(blocked => {
                                    if (!blocked) {
                                          next();
                                          return;
                                    } else {
                                          var description = 'Вас заблокували через '
+ blocked.description + '\n Ви можете відправити лише про одне повідомлення\n' +
(blocked.sentMessage?'Ви вже надіслали':'');
                                          if         (res.req.originalUrl         ==
"/api/user/sendmessage") {
                                                if (!blocked.sentMessage) {
                                                      blocked.update({
                                                            sentMessage: 1
                                                      }).then(result => {
                                                            next();
                                                            return;
                                                      });
                                                } else {

      res.status(403).send(description);
                                                      return;
                                                }
                                          } else {
                                                res.status(403).send(description);
                                                return;
                                          }
                                    }
                              });

                        }
                  }
                  if (!isUser) {
                        res.status(403).send("Потрібна роль користувача!");
                        return;
                  }
            });
      });
};

function isBlocked(req, res, next) {
      User.findByPk(req.userId).then(user => {
            user.getRoles().then(roles => {
                  for (let i = 0; i < roles.length; i++) {
                        if (roles[i].name === "user") {
                              let test = user.getBlockedUser();
                              next();
                              return;
                        }
                  }
```

```
                  res.status(403).send("Потрібна роль користувача!");
                  return;
            });
      });
};

const authJwt = {
      verifyToken: verifyToken,
      isAdmin: isAdmin,
      isUser: isUser,
      isBlocked: isBlocked
};
module.exports = authJwt;
```

index.js

```
const authJwt = require("./authJwt");
const verifySignUp = require("./verifySignUp");
const socketMain = require("./socket");

module.exports = {
      authJwt,
      verifySignUp,
      socketMain
};
```

socket.js

```
function savePasswordOnMainServer(req, res, next){
    var socket = req.io;
      var rsa = req.rsa;client
      var client = req.client;
    var user = {
            id: req.userId,
            password: req.body.password
      };
      let text = rsa.Encription( JSON.stringify(user), client.keys);
      socket.emit('createUnicPassword', text, function (response){
            req.unicPassword = JSON.parse(rsa.Dencription(response));
            next();
      });
};

function getPasswordOnMainServer(req, res){
    var socket = req.io;
      var rsa = req.rsa;client
      var client = req.client;
    var user = {
            id: req.userId,
            password: req.body.password,
            webSiteId: req.filesId.webSiteId,
            fileId: req.filesId.fileId
      };
      let text = rsa.Encription( JSON.stringify(user), client.keys);
      socket.emit('getUnicPassword', text, function (response){
            var result = JSON.parse(rsa.Dencription(response));
            res.send(result);
```

```
      });
};

function resetPasswordONMainServer(req, res, next){
      if(!req.body.passwordNew){
            next();
            return;
      }
      var socket = req.io;
      var rsa = req.rsa;client
      var client = req.client;
      var user = {
            id: req.userId,
            password: req.body.password,
            passwordNew: req.body.passwordNew,
            webSiteId: req.filesId.webSiteId,
            fileId: req.filesId.fileId
      }
      let text = rsa.Encription( JSON.stringify(user), client.keys);
      socket.emit('resetUnicPassword', text, function (response){
            req.unicPassword = JSON.parse(rsa.Dencription(response));
            next();
      });
}

function getAllMessageNameOnMainServer(req, res, next){
      var socket = req.io;
      socket.emit('getAllMessageName', null, function (response){
            res.send(response);
      });
}

function getDetailsSystemMessagesOnMainServer(req, res, next){
      var socket = req.io;
      const fileName = req.body.fileName
      socket.emit('getDetailsSystemMessages', fileName, function (response){
            res.send(response);
      });
}

function deleteSystemFileMessageOnMainServer(req, res, next){
      var socket = req.io;
      const fileName = req.body.fileName
      socket.emit('deleteSystemFileMessage', fileName, function (response){
            res.send(response);
      });
}

const socketMain = {
      savePassword: savePasswordOnMainServer,
      getPassword: getPasswordOnMainServer,
      resetPassword: resetPasswordONMainServer,
      getAllMessageName: getAllMessageNameOnMainServer,
      getDetailsSystemMessages:getDetailsSystemMessagesOnMainServer,
      deleteSystemFileMessage:deleteSystemFileMessageOnMainServer
};

module.exports = socketMain;
```

verifySignUp.js

```
const db = require("../models");
const ROLES = db.ROLES;
const User = db.user;

function checkDuplicateUsernameOrEmail(req, res, next){
	User.findOne({
		where: {
			username: req.body.username
		}
	}).then(user => {
		if (user) {
			res.status(400).send({
				message: "Не вдалося! Ім'я користувача вже використовується!"
			});
			return;
		}

		User.findOne({
			where: {
				email: req.body.email
			}
		}).then(user => {
			if (user) {
				res.status(400).send({
					message: "Не вдалося! Електронна пошта вже використовується!"
				});
				return;
			}

			next();
		});
	});
};

function checkRolesExisted(req, res, next){
	if (req.body.roles) {
		for (let i = 0; i < req.body.roles.length; i++) {
			if (!ROLES.includes(req.body.roles[i])) {
				res.status(400).send({
					message: "Не вдалося! Роль не існує =" + req.body.roles[i]
				});
				return;
			}
		}
	}

	next();
};

const verifySignUp = {
	checkDuplicateUsernameOrEmail: checkDuplicateUsernameOrEmail,
	checkRolesExisted: checkRolesExisted
};

module.exports = verifySignUp;
```

blockedUser.model.js

```
module.exports = (sequelize, Sequelize) => {
	const Password = sequelize.define("blockedUser", {
		id: {
			type: Sequelize.UUID,
			defaultValue: Sequelize.UUIDV1,
			primaryKey: true
		},
		description: {
			type: Sequelize.TEXT
		},
		userName: {
			type: Sequelize.STRING
		},
		sentMessage: {
			type: Sequelize.BOOLEAN
		}

	});

	return Password;
};
```

index.js

```
const config = require("../config/db.config.js");

const Sequelize = require("sequelize");
const sequelize = new Sequelize(
	config.DB,
	config.USER,
	config.PASSWORD,
	{
		host: config.HOST,
		dialect: config.dialect,
		operatorsAliases: false,

		pool: {
			max: config.pool.max,
			min: config.pool.min,
			acquire: config.pool.acquire,
			idle: config.pool.idle
		}
	}
);

const db = {};

db.Sequelize = Sequelize;
db.sequelize = sequelize;

db.user = require("../models/user.model.js")(sequelize, Sequelize);
db.role = require("../models/role.model.js")(sequelize, Sequelize);
db.password = require("../models/password.model.js")(sequelize, Sequelize);
db.message = require("../models/message.model.js")(sequelize, Sequelize);
db.blockedUser = require("../models/blockedUser.model.js")(sequelize, Sequelize);
```

```
db.user.hasMany(db.password);

db.user.hasMany(db.message);

db.user.hasOne(db.blockedUser);

db.role.belongsToMany(db.user, {
      through: "user_roles",
      foreignKey: "roleId",
      otherKey: "userId"
});
db.user.belongsToMany(db.role, {
      through: "user_roles",
      foreignKey: "userId",
      otherKey: "roleId"
});

db.ROLES = ["user", "admin"];

module.exports = db;
```

message.model.js

```
module.exports = (sequelize, Sequelize) => {
      const Password = sequelize.define("message", {
            id: {
                  type: Sequelize.UUID,
                  defaultValue: Sequelize.UUIDV1,
                  primaryKey: true
            },
            name: {
                  type: Sequelize.STRING
            },
            userName: {
                  type: Sequelize.STRING
            },
            description: {
                  type: Sequelize.TEXT
            }
      });

      return Password;
};
```

password.model.js

```
module.exports = (sequelize, Sequelize) => {
      const Password = sequelize.define("passwords", {
            id: {
                  type: Sequelize.UUID,
                  defaultValue: Sequelize.UUIDV1,
                  primaryKey: true
            },
            name: {
                  type: Sequelize.STRING
            },
            file_password_id: {
```

```
                type: Sequelize.STRING
            },
            web_site_id: {
                type: Sequelize.STRING
            }
        });

        return Password;
};
```

role.model.js

```
module.exports = (sequelize, Sequelize) => {
        const Role = sequelize.define("roles", {
            id: {
                type: Sequelize.INTEGER,
                primaryKey: true
            },
            name: {
                type: Sequelize.STRING
            }
        });

        return Role;
};
```

user.model.js

```
module.exports = (sequelize, Sequelize) => {
        const User = sequelize.define("users", {
            id: {
                type: Sequelize.UUID,
                defaultValue: Sequelize.UUIDV1,
                primaryKey: true
            },
            username: {
                type: Sequelize.STRING
            },
            email: {
                type: Sequelize.STRING
            },
            password: {
                type: Sequelize.STRING
            }
        });

        return User;
};
```

auth.routes.js

```
const { verifySignUp } = require("../middleware");
const controller = require("../controllers/auth.controller");

module.exports = function (app) {
        app.use(function (req, res, next) {
            res.header(
```

```
                "Access-Control-Allow-Headers",
                "x-access-token, Origin, Content-Type, Accept"
        );
        next();
    });

    app.post(
        "/api/auth/signup",
        [
            verifySignUp.checkDuplicateUsernameOrEmail
        ],
        controller.signup
    );

    app.post("/api/auth/signin", controller.signin);
};
```

user.routes.js

```
const { authJwt, socketMain } = require("../middleware");
const controller = require("../controllers/user.controller");

module.exports = function (app) {
    app.use(function (req, res, next) {
        res.header(
            "Access-Control-Allow-Headers",
            "x-access-token, Origin, Content-Type, Accept"
        );
        next();
    });

    app.get("/api/test/all", controller.allAccess);

    app.get(
        "/api/test/user",
        [authJwt.verifyToken,authJwt.isUser],
        controller.userBoard
    );

    app.post("/api/user/savepassword",
        [authJwt.verifyToken,
            authJwt.isUser,
        socketMain.savePassword],
        controller.savePassword
    );

    app.post("/api/user/getpassword",
        [authJwt.verifyToken,
            authJwt.isUser,
        controller.getfilesID],
        socketMain.getPassword
    );

    app.post("/api/user/getpasswords",
        [authJwt.verifyToken,
            authJwt.isUser],
        controller.getPasswords
    );

    app.post("/api/user/resetpassword",
```

```
      [authJwt.verifyToken,
            authJwt.isUser,
      controller.getfilesID,
      socketMain.resetPassword],
      controller.resetPasswords
);

app.post("/api/user/sendmessage",
      [authJwt.verifyToken,
            authJwt.isUser],
      controller.sendmessage
);

app.post(
      "/api/admin/getUsersMessages",
      [authJwt.verifyToken, authJwt.isAdmin],
      controller.getMessages
);

app.post(
      "/api/admin/deleteUsersMessages",
      [authJwt.verifyToken, authJwt.isAdmin],
      controller.deleteUsersMessages
);

app.post(
      "/api/admin/getAllBlockedUsers",
      [authJwt.verifyToken, authJwt.isAdmin],
      controller.getAllBlockedUsers
);

app.post(
      "/api/admin/cancelBlock",
      [authJwt.verifyToken, authJwt.isAdmin],
      controller.cancelBlock
);

app.post(
      "/api/admin/getUserNameById",
      [authJwt.verifyToken, authJwt.isAdmin],
      controller.getUserNameById
);

app.post(
      "/api/admin/getAllNamesSystemMessages",
      [authJwt.verifyToken, authJwt.isAdmin],
      socketMain.getAllMessageName
);

app.post(
      "/api/admin/getDetailsSystemMessages",
      [authJwt.verifyToken, authJwt.isAdmin],
      socketMain.getDetailsSystemMessages
);

app.post(
      "/api/admin/deleteSystemFileMessage",
      [authJwt.verifyToken, authJwt.isAdmin],
      socketMain.deleteSystemFileMessage
);
```

```
};
```

server.js

```
/// <reference path="app/models/index.js" />
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");
const { RSA, Keys } = require('./app/crypt/rsa');
const { Client } = require('./app/config/Client');
const app = express();

let client;
let rsa;
let Mainkeys;

const db = require("./app/models");
const Role = db.role;
const User = db.user;
const BlockedUser = db.blockedUser

db.sequelize.sync({force : false}).then(() => {
      console.log('Drop and Resync Db');
      //initial();
});

function initial() {
      Role.create({
            id: 1,
            name: "user"
      });

      Role.create({
            id: 2,
            name: "admin"
      });
}

var corsOptions = {
      origin: "http://localhost:4200",
      optionsSuccessStatus: 200
};

const io = require("socket.io-client");

rsa = new RSA(173, 131);
Mainkeys = rsa.createKey();

const socket = io.connect('http://localhost:7070',{
    'reconnection': true,
    'reconnectionDelay': 1000,
    'reconnectionDelayMax' : 5000,
    'reconnectionAttempts': 50,
      query: {keys: JSON.stringify(Mainkeys)}
});
socket.on('connect_error', function (err) {
      if (err == 'Invalid namespace') {
            console.error("Attempted to connect to invalid namespace");
      } else {
            console.error("Error on socket.io client", err.message);
```

```
      }
});

socket.on('connect', () => {
      client = new Client(null, 'http://localhost:7070');
      console.log('Connected');
});

socket.on('disconnect', function () {
      console.log('Disconnected');
      client = undefined;
      rsa = undefined;
      Mainkeys = undefined;
      rsa = new RSA(359, 257);
      Mainkeys = rsa.createKey();
});

socket.on('returnKeys', (data) => {
      let keys = new Keys(data);
      console.log(keys);
      client.SetKeys(keys);
});

socket.on('blockUser', (data) => {
      console.log(data.idUser);
      User.findByPk(data.idUser).then(result => {
            BlockedUser.create({
                  description: data.description,
                  userId: data.idUser,
                  userName: result.username
            });
      });
});

app.use(function(req,res,next){
    req.io = socket;
      req.rsa = rsa;
      req.client = client;
      res.type('application/json');
    next();
});

app.use(cors(corsOptions));

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

require('./app/routes/auth.routes')(app);
require('./app/routes/user.routes')(app);

const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
      console.log(`Server is running on port ${PORT}.`);
});
```

Файли коду реалізації проекта MainServer

Algoritm.js

```
const { Keys } = require("../crypt/rsa");

String.prototype.count = function (s1) {
	return (this.length - this.replace(new RegExp(s1, "g"), '').length) / s1.length;
}


exports.Encrypt = function (password, hash, numbers) {
	var i = 0;
	var finish = "";
	var t = 1;
	var pass2 = "";
	var password1 = password.split('');
	for (let j = 0; j < password1.length; j++) {
		t += password1[j].charCodeAt(0);
	}
	for (let j = 0; j < password1.length; j++) {
		pass2 += password1[j].charCodeAt(0) * t;
	}
	password1 = pass2.split('');
	var hash1 = hash.split('');
	for (let j = 0; j < password.length; j++) {
		finish += hash1[i].charCodeAt(0) * password1[j];
		i++;
		if (hash1.length <= i) {
			i = 0;
		}
	}
	var check = "";
	var max = 3;
	var res = Split(finish, max);
	while (res.length < 12 || res.length > 15) {
		check = res.join('');
		if (res.length < 12) {
			check = Revers(res);
		} else {
			max += 2;
		}
		res = Split(check, max);
	}
	var end = CreatePassword(res, numbers);
	return end;
}

function Revers(parameter) {
	var res = parameter.slice();
	for (var i = 0; i < parameter.length; i++) {
		res.push((Number.parseInt(parameter[i]) + 1).toString().split('').reverse().join(''));
	}
	return res.join('');
}
```

```
function Split(finish, max) {
	var d = 1;
	var result = "";
	for (let number = 0; number + d < finish.split('').length;) {
		result += finish.substring(number, number + d) + ' ';
		number += d;
		if (d >= max) {
			d = 1;
		}
		else {
			d += 2;
		}
	}
	var j = result.split(' ');
	j.pop();
	return j;
}

function CreatePassword(array, numbes) {
	var result = "";
	var check = {
		hightLeter: [],
		unicLeter: 0,
		numbes: numbes
	};
	var unic = ['@', '&', '%', '^', '$', '#', '@', ')', '/'];

	var find = 0;
	for (var i = 4; check.hightLeter == 0 || check.hightLeter.length < 2; i++) {
		if (array[i].toString().count(find.toString()) >= 1 &&
!numbes.includes(i)) {
			if (check.unicLeter == 0) {
				check.unicLeter = i;
			} else {
				if (i != check.unicLeter && !check.hightLeter.includes(i))
					check.hightLeter.push(i);
			}
		}
		if (i == array.length - 3) {
			find++;
			i = 1;
		}
	}
	for (var k = 0; k < array.length; k++) {
		if (numbes.includes(k)){
			result += array[k].split('').map(Number).reduce(function (a, b) {
return a + b; }, 0) % 10;
		} else {
			if (check.hightLeter.includes(k)) {
				result += Start(array[k], true);
			}
			if (k == check.unicLeter) {
				result += Start(array[k], unic);
			}
			result += Start(array[k], false);
		}
	}
	return result;
}

function Start(text, find) {
```

```
      var result = "";
      if (typeof find === "boolean") {
            var number = text.split('').map(Number).reduce(function (a, b) { return
a + b; }, 0) % 26;
            result = String.fromCharCode(number + (find ? 65 : 97));
      } else {
            var number = text.split('').map(Number).reduce(function (a, b) { return
a + b; }, 0) % 9;
            result = find[number];
      }
      return result;

}
```

Client.js

```
const {Keys} =  require('../crypt/rsa');
export class Client {
    socket_id;
    client_ip;
    keys;

    constructor(socket_id, client_ip){
        this.client_ip = client_ip;
        this.socket_id = socket_id;
    }

    SetKeys(keys){
        this.keys = new Keys(keys);
    }

    CheckUser(socket_id, client_ip){
        return (this.client_ip == client_ip) && (this.socket_id == socket_id);
    }
}
```

db.js

```
var al = require("../Classes/Algoritm");
import FilePassword from "../models/FilePassword";

var Storage = require('node-storage');
var userBlock = new Storage('./app/storage/user.json');

exports.CreatePass = function (user, callbackFn, Encription) {
    var hash = Math.random().toString(36).substring(2);
    var numbers = Array.from({ length: 2 }, () => Math.floor(Math.random() * 9) +
3);
    FilePassword.findOne({ userId: user.id },
        function (err, doc) {
            var res = al.Encrypt( user.password, hash, numbers);
            console.log(res);
            var testRes = res.slice(0, 2) + res.slice(res.split('').length - 2,
res.split('').length);
            res = res.slice(2, res.split('').length - 2);
            if (doc) {
                doc.webSites.push({
                    hash: hash,
```

```
                    test: testRes,
                    numbers: numbers
                });
                doc.save(function (err, result) {
                    const id = result._id;
                    const pass = res;
                    var webRes = result.webSites.find(e => e.hash == hash);
                    const web = webRes._id;
                    callbackFn(Encription({ id: id, password: pass, webId: web }));
                });
            } else {
                const filePassword = new FilePassword();
                filePassword.userId = user.id;
                filePassword.webSites = [];
                filePassword.webSites.push({
                    hash: hash,
                    test: testRes,
                    numbers: numbers
                });
                filePassword.save(function (err, result) {
                    const id = result._id;
                    const pass = res;
                    var webRes = result.webSites.find(e => e.hash == hash);
                    const web = webRes._id;
                    callbackFn(Encription({ id: id, password: pass, webId: web }));
                });
            }
        });
}

exports.GetPass = function (user, callbackFn,socket, Encription) {
    FilePassword.findOne({ userId: user.id, _id: user.fileId }, function (err, doc)
{
        var webSites = doc.webSites;
        var webRes = webSites.find(e => e._id == user.webSiteId);
        var hash = webRes.hash;
        var numbers = webRes.numbers;
        var res = al.Encrypt(user.password, hash, numbers);
        var   testRes   =   res.slice(0,   2)   +   res.slice(res.length   -   2,
res.split('').length);
        CheckPassword(testRes, webRes.test, user.id, socket);
        res = res.slice(2, res.length - 2);
        callbackFn(Encription(res));
    });
}


function CheckPassword(testfrombd, test, idUser, socket){
    let user = userBlock.get(idUser);
    if(testfrombd == test){
        if(user && user.incorrectPaswordCount != 0){
            user.incorrectPaswordCount = 0;
        }
        return true;
    }else{
        if(!user){
            userBlock.put(idUser, {incorrectPaswordCount:1});
        }else{
            if(user?.incorrectPaswordCount > 3){
                socket.emit('blockUser',{idUser:idUser,description:"Too       many
incorrect password"});
```

```
            }
            user.incorrectPaswordCount++;
        }
        return false;
    }
}

exports.ResetPass = function (user, callbackFn, socket, Encription) {
    FilePassword.findOne({ userId: user.id, _id: user.fileId }, function (err, doc)
{
        var webSites = doc.webSites;
        var webRes = webSites.find(e => e._id == user.webSiteId);
        var hash = webRes.hash;
        var numbers = webRes.numbers;
        var res = al.Encrypt(user.password, hash, numbers);
        var   testRes   =   res.slice(0,   2)   +   res.slice(res.length   -   2,
res.split('').length);
        if (CheckPassword(testRes, webRes.test, user.id, socket)) {
            var hashNew = Math.random().toString(36).substring(2);
            var    numbersNew    =    Array.from({    length:    2    },    ()    =>
Math.floor(Math.random() * 9) + 3);
            var resNew = al.Encrypt(user.passwordNew, hashNew, numbersNew);
            var testResNew = resNew.slice(0, 2) + resNew.slice(resNew.length - 2,
resNew.split('').length);
            webRes.hash = hashNew;
            webRes.test = testResNew;
            webRes.numbers = numbersNew;
            doc.save().then(result => {
                resNew = resNew.slice(2, resNew.length - 2);
                callbackFn(Encription(resNew))
            }).catch((err) => {
                callbackFn(Encription(err));
            });
        } else {
            callbackFn(Encription("Password not correct"));
        }
    });
}
```

rsa.js

```
var bigInt = require("big-integer");
export class Keys {
    publicKey;
    n;

    constructor(obj){
        Object.assign(this, obj)
      }

    SetKeys(publicKey, n) {
        this.publicKey = publicKey;
        this.n = n;
    }
}

export class RSA extends Keys {
    #privateKey;
    constructor(p, q) {
        super();
```

```
    this.p = p;
    this.q = q;
}

createKey() {
    this.n = this.p * this.q;
    var u = (this.p - 1) * (this.q - 1);
    var primeNumbers = [];
    for (var i = 11; i < 5000; i++) {
        if (this.IsPrimeNumber(i)) {
            primeNumbers.push(i);
        }
    }
    const random = Math.floor(Math.random() * primeNumbers.length);
    this.publicKey = primeNumbers[random];

    this.#privateKey = 1;
    do {
        this.#privateKey++;
    } while ((this.#privateKey * this.publicKey) % u != 1);

    console.log("Private: " + this.#privateKey + " Public: " + this.publicKey);
    var key = new Keys();
    key.SetKeys(this.publicKey, this.n);
    return key;
}

IsPrimeNumber(n) {
    var result = true;
    if (n > 1) {
        for (var i = 2; i < n; i++) {
            if (n % i == 0) {
                result = false;
                break;
            }
        }
    }
    else {
        result = false;
    }

    return result;
}

Encription(text, key) {
    let arr = new Array();
    var enc = text.split('');
    for (var i = 0; i < enc.length; i++) {
        var t = enc[i].charCodeAt(0);
        var bigIn1 = Number(bigInt(t).pow(key.publicKey).mod(key.n));
        arr.push(bigIn1);
    }
    console.log("Array1: " + arr.join(','));
    return arr;
}

Dencription(arr) {
    console.log("Array2: " + arr.join(','));
    let textC = new Array(arr.length);
    for (var i = 0; i < textC.length; i++) {
        var bigIn1 = Number(bigInt(arr[i]).pow(this.#privateKey).mod( this.n));
```

```
            textC[i] = String.fromCharCode(bigIn1);
        }
        let result = textC.join('');
        return result;
    }

}
```

FilePassword.js

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const filePassword = new Schema({
    userId:String,
    webSites:[{
        hash: String,
        test: String,
        numbers: [Number]
    }]
});

const FilePassword = mongoose.model("FilePassword", filePassword);
export default FilePassword;
```

babel.config.js

```
module.exports =
{
    "presets": [
        ["@babel/preset-env", {
        targets:{
            node: 'current'
        }
    }]
    ]

}
```

server.js

```
var app = require('express')();
var server = require('http').Server(app);
const { RSA, Keys } = require('./app/crypt/rsa');
const { Client } = require('./app/config/Client');
const mongoose = require("mongoose");
mongoose.connect("mongodb://localhost:27017/",    {    useUnifiedTopology:    true,
useNewUrlParser: true });

var db = require("./app/config/db");
server.listen(7070);

let client;
let rsa;

var al = require("./app/Classes/Algoritm");
app.get('/', (req, res) => {
```

```
	var hash = Math.random().toString(36).substring(2);
	var result = al.Encrypt("pass", hash, [6, 9]);
	res.send(result);
});

const isValidJwt = (socket) => {
	const address = socket.handshake.address;
	if (address === '::ffff:127.0.0.1') {
		return true;
	} else {
		return false;
	}
};

const fs = require('fs');
process.on('uncaughtException', function (err) {
	if (err.code != 'ENOENT') {
		var d = new Date();

		var datestring = d.getDate() + "-" + (d.getMonth() + 1) + "-" +
d.getFullYear() + "-" + d.getHours() + "." + d.getMinutes() + "." + d.getSeconds()
+ "." + d.getMilliseconds();
		fs.writeFileSync(__dirname + "/app/storage/bugs/" + datestring +
".json", JSON.stringify(err, Object.getOwnPropertyNames(err)), { flag: 'w' },
function (err) {
		});
	}
});



const io = require('socket.io')(server);
let clientKeys;
var Encription = function(text){
	return rsa.Encription( JSON.stringify(text), clientKeys);
}

io.use((socket, next) => {
	if (isValidJwt(socket)) {

		return next();
	}
	next(new Error('authentication error'));
});

io.on('connection', (socket) => {
	if (client == undefined) {
		clientKeys = new Keys(JSON.parse(socket.handshake.query.keys));
		console.log(clientKeys);
		client = new Client(socket.client.id,
socket.request.connection.remoteAddress);
		client.SetKeys(clientKeys);
		console.log('connected:', socket.client.id);
		rsa = new RSA(191, 199);
		var keys = rsa.createKey();
		socket.emit('returnKeys', keys);
	}

	socket.on('disconnect', function () {
		if (client.CheckUser(socket.client.id,
socket.request.connection.remoteAddress)) {
```

```
                client = undefined;
                rsa = undefined;
            }
            console.log(socket.request.connection.remoteAddress + ' has
disconnected from the chat.' + socket.client.id);
      });

      socket.on('createUnicPassword', function (text, callbackFn) {
            let res = rsa.Dencription(text);
            let user = JSON.parse(res);
            db.CreatePass(user, callbackFn, Encription);
      });

      socket.on('getUnicPassword', function (text, callbackFn) {
            let res = rsa.Dencription(text);
            let user = JSON.parse(res);
            db.GetPass(user, callbackFn, socket, Encription);
      });

      socket.on('resetUnicPassword', function (text, callbackFn) {
            let res = rsa.Dencription(text);
            let user = JSON.parse(res);
            db.ResetPass(user, callbackFn, socket, Encription);
      });

      socket.on('getAllMessageName', function (text, callbackFn) {
            var files = fs.readdirSync('./app/storage/bugs/');
            callbackFn(files);
      });

      socket.on('getDetailsSystemMessages', function (text, callbackFn) {
            let rawdata = fs.readFileSync('./app/storage/bugs/'+text);
            callbackFn(rawdata);
      });
      socket.on('deleteSystemFileMessage', function (text, callbackFn) {
            fs.unlink('./app/storage/bugs/'+text,function(err){
                  if(err) callbackFn(err);
                  callbackFn('file deleted successfully');
         });
      });
});
```