Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Бакалаврська дипломна робота на тему:
«Засіб для керування паролями на основі апаратного токена»

Виконав: студент 4 курсу групи 1БС-20б
спеціальності 125 Кібербезпека
_____ Сергій ГУРІН

Керівник: к. т. н., доц., доц. каф. ЗІ
_____ Олеся ВОЙТОВИЧ
«14» червня 2024 р.

Рецензент: к. т. н. доцент каф. ПЗ
_____ Ганна РАКИТЯНСЬКА
«14» червня 2024 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«14» червня 2024 р.

Вінниця ВНТУ – 2024 року

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Войтович О.П., к.т.н., доц. каф. ЗІ.	11.03.24	31.03.24
2	Войтович О.П., к.т.н., доц. каф. ЗІ.	11.03.24	15.04.24
3	Войтович О.П., к.т.н., доц. каф. ЗІ.	11.03.24	01.05.24

7. Дата видачі завдання 12 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Прим
1	Аналіз завдання. Вступ	12.03.24 – 14.03.24	
2	Аналіз інформаційних джерел за напрямком бакалаврської дипломної роботи	15.03.24 – 31.03.24	
3	Розробка рішень, моделей, алгоритмів	01.04.24 – 15.04.24	
4	Практична реалізація, моделювання, експериментування, результати	16.04.24 – 01.05.24	
5	Оформлення пояснювальної записки	02.05.24 – 10.05.24	
6	Попередній захист БДР	30.05.24 – 07.06.24	
7	Виправлення зауважень, підготовка ілюстративного матеріалу	10.06.24 – 12.06.24	
8	Перевірка на наявність текстових запозичень	12.06.24 – 13.06.24	
9	Представлення БДР до захисту, рецензування	13.06.24 – 17.06.24	
10	Захист БДР	18.06.24 – 21.06.24	

Студент _____ Сергій ГУ
(підпис)

Керівник роботи _____ Олеся ВОЙТОВ
(підпис)

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти – І (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
зав. кафедри ЗІ, д. т. н., проф.
Володимир ЛУЖЕЦЬКИЙ
12 березня **2024 року**

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Гуріну Сергію В'ячеславовичу

1. Тема роботи: «Засіб для керування паролями на основі апаратного токена» керівник роботи: Войтович Олеся Петрівна, к.т.н., доц., доц. каф. ЗІ. затверджені наказом ректора ВНТУ від 11 березня 2024 року №80.
2. Строк подання студентом роботи: 14 червня 2024 року.
3. Вихідні дані до роботи:
   –зберігання паролів з використанням апаратного токена;
   –модуль реєстрації за допомогою апаратного токена;
   – модуль автентифікації за допомогою апаратного токена.
4. Зміст розрахунково-пояснювальної записки:
   Вступ. Аналіз предметної області. Проектування системи. Розробка програмного засобу та експериментальні дослідження. Висновки. Перелік використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Загальна структура програмного засобу (Плакат, А4), Загальна структура розробленого програмного засобу та зв'язків між класами (Плакат, А4), Схема роботи алгоритму зчитування інформації з апаратного токену, (Плакат, А4), Схема передачі запитів на виконання операцій читання, збереження, оновлення та видалення записів до бази даних (Плакат, А4), Схема роботи алгоритму генерації паролів (Плакат, А4), Схема алгоритму генерації секретного ключа шифрування (Плакат, А4), Схема роботи алгоритму оновлення майстер-паролю та прив'язки до нового апаратного токена, результати тестування (Плакат, А4).

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 82 сторінок формату А4, на яких є 39 рисунків, 2 таблиці, список використаних джерел містить 45 найменувань.

Бакалаврська робота присвячена розробці програмного засобу керування паролями на основі апаратного токену. Проведено аналіз існуючих технологій, що використовуються в апаратних токенах та сучасних засобів керування паролями, що дозволило ідентифікувати ключові характеристики та вимоги для розробки власного рішення. На основі проведеного аналізу було розроблено програмний засіб керування паролями, який забезпечує можливість автентифікації та реєстрації користувачів за допомогою USB-носія. Для розробки було використано мову програмування Java, що дозволило реалізувати необхідні функції та забезпечити високу надійність системи. Розроблений засіб пройшов тестування, що включало перевірку на коректність роботи алгоритмів автентифікації та збереження даних. Результати тестування підтвердили ефективність та надійність створеного програмного забезпечення, що робить його перспективним для використання у практичних умовах.

Ключові слова: апаратний токен, технології апаратних токенів, реєстрація, автентифікація, логін, пароль, облікові записи, зберігання даних, безпека даних, програмне забезпечення, USB-носій, управління обліковими записами, тестування програмного забезпечення, шифрування даних.

ABSTRACT

The bachelor's thesis consists of 82 pages of A4 format, with 39 figures, 2 tables, and a list of references containing 45 titles.

The bachelor's thesis is devoted to the development of a password management software tool based on a hardware token. An analysis of existing technologies used in hardware tokens and modern password management tools was conducted, which allowed identifying key characteristics and requirements for developing an in-house solution. Based on this analysis, a software password management tool was developed that provides the ability to authenticate and register users using a USB key. The Java programming language was used for the development, which made it possible to implement the necessary functions and ensure high reliability of the system. The developed tool was tested, including checking the correctness of the authentication and data storage algorithms. The test results confirmed the effectiveness and reliability of the created software, which makes it promising for use in practical conditions.

Keywords: hardware token, hardware token technology, registration, authentication, login, password, accounts, data storage, data security, software, USB key, account management, software testing, data encryption.

ЗМІСТ

ВСТУП

В умовах сучасного інформаційного суспільства безпека даних стає однією з найважливіших складових функціонування будь-якої організації та забезпечення приватності особистої інформації користувачів. Одним із ключових елементів системи захисту інформації є паролі, які виступають як засоби автентифікації користувачів при доступі до різноманітних ресурсів та послуг. Однак, через зростаючу кількість онлайн-акаунтів та складність запам'ятовування великої кількості паролів, користувачі часто стикаються з проблемами їх зберігання та використання.

Сучасні менеджери паролів поділяються на програмні та апаратні рішення. Програмні менеджери паролів зберігають зашифровані паролі на локальному пристрої користувача або у хмарному сховищі, забезпечуючи зручний доступ до них через додатки чи браузерні розширення. Водночас вони залишаються вразливими до атак зловмисників, шкідливого ПЗ та можливих витоків даних. Апаратні менеджери паролів використовують фізичні токени, що забезпечують додатковий рівень безпеки завдяки апаратному шифруванню та автентифікації. Розробка менеджера паролів з автентифікацією на основі апаратного токену є надзвичайно актуальною, оскільки поєднує зручність використання програмних рішень з високим рівнем безпеки апаратних токенів, мінімізуючи ризик компрометації паролів навіть у разі фізичного доступу до пристрою чи спроб фішингових атак. Тема бакалаврської роботи є актуальною у контексті сучасних вимог до безпеки інформаційних систем.

Об’єктом бакалаврської дипломної роботи є процеси автентифікації користувачів та зберігання паролів.

Предметом бакалаврської дипломної роботи є методи та засоби автентифікації на основі апаратного токена.

Метою бакалаврської кваліфікаційної роботи є покращення керування паролями у інформаційно-комунікаційних системах та підвищення безпеки зберігання даних, на основі розробки та реалізації засобу для керування паролями,

що базується на використанні апаратного токена, який забезпечить високий рівень захисту та зручність використання для кінцевих користувачів.

Для досягнення цієї мети необхідно вирішити такі **завдання**:

– провести аналіз існуючих засобів керування паролями, їх переваг та недоліків;

– дослідити сучасні технології та стандарти;

– реалізувати механізми реєстрації та автентифікації за допомогою апаратного токену;

– забезпечити шифрування даних для збереження їхньої конфіденційності та захисту від несанкціонованого доступу;

– розробити архітектуру програмного засобу;

– розробити програмний засіб;

– провести тестування програмного засобу та оцінити його ефективність.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Значення паролів у сучасних інформаційних системах

Паролі є основним засобом автентифікації користувачів у сучасних інформаційних системах. Вони є першим рубежем захисту від несанкціонованого доступу до облікових записів, конфіденційної інформації та критично важливих даних [1].

Попри розвиток біометричних та інших сучасних методів [2], паролі залишаються популярними завдяки їх простоті, зручності та відносній ефективності. Користувачі створюють паролі для доступу до своїх акаунтів, систем, мереж і різних ресурсів, що забезпечує контрольований доступ і захист даних від несанкціонованого використання.

З розвитком Інтернету та технологій комп'ютери стали незамінним обладнанням для будь-якої роботи. Як наслідок, сьогодні люди щодня користуються багатьма обліковими записами, що призводить до необхідності запам'ятовувати більшу кількість паролів. Говорячи про людей з більш складною комп'ютерною роботою, наприклад, про тих, хто працює в IT-індустрії, кількість паролів, які вони використовують щодня, може бути досить великою.

Незважаючи на свою важливість, паролі мають свої вразливості. Однією з основних проблем є використання слабких або повторюваних паролів, які легко вгадати чи підібрати. Використання одних і тих самих паролів на багатьох акаунтах є надзвичайно небезпечним, оскільки компрометація одного облікового запису автоматично ставить під загрозу всі інші облікові записи з тим самим паролем [3]. Це означає, що якщо зловмисник отримає доступ до одного з акаунтів, він зможе легко отримати доступ до всіх інших, використовуючи той самий пароль. Така практика значно підвищує ризик масового витоку особистих даних і зловживання обліковими записами. Щоб забезпечити належний рівень безпеки, важливо використовувати унікальні, складні паролі для кожного облікового запису. Крім того, користувачі часто нехтують рекомендаціями щодо регулярної зміни паролів і їхнього зберігання в безпечних місцях, записуючи їх на папері, або зберігаючи у відкритому вигляді, наприклад, у нотатках чи галереї смартфону, що негативно

впливає на конфіденційність таких даних і може призвести до компрометації облікових записів. Паперові записи можуть бути загублені або знайдені іншими людьми, що ставить конфіденційні дані під загрозу, а додатки для нотаток і галереї на смартфонах часто не мають достатнього рівня захисту, що робить їх уразливими до хакерських атак. Якщо ж смартфон потрапить до рук зловмисника або буде втрачений, всі збережені у відкритому вигляді паролі будуть легко доступними, що може призвести до несанкціонованого доступу до облікових записів користувача та особистої інформації. Незважаючи на наявність біометричного захисту або PIN-коду на смартфоні, зловмисники можуть використовувати різні методи для отримання доступу до конфіденційної інформації, такої як файли, повідомлення та інші дані. Це може відбутися через, наприклад, встановлення шкідливих додатків на смартфон може відбутися через завантаження заражених файлів або використання небезпечних Wi-Fi мереж. Після установки, шкідливе ПЗ може отримати привілейований доступ до системи, обійти біометричний захист і отримати доступ до конфіденційних файлів [4].

Використання менеджерів паролів та багатофакторної автентифікації допомагає вирішувати ці проблеми [5]. Вони генерують надійні паролі, зберігають їх і захищають за допомогою шифрування. Перевага полягає в тому, що користувачу ніколи не доведеться запам'ятовувати паролі, навіть якщо у нього їх сотні. Окрім того, існує можливість синхронізації паролів на всіх пристроях, що забезпечує доступ з будь-якого пристрою в будь-який час. Паролі можна отримати за потреби, і немає необхідності їх запам'ятовувати. Також зловмисники можуть обманним шляхом змусити користувача розкрити свій PIN-код або біометричні дані через фальшиві додатки, електронні листи чи повідомлення. Використовуючи соціальну інженерію, вони можуть переконати жертву виконати певні дії, які розкриють захисні дані.

Аналізуючи можливі загрози, не виникає сумнівів, що необхідність створення надійного програмного засобу для безпечного зберігання паролів є досить важливою. Розробка менеджера паролів на основі апаратного токену значно підвищує рівень безпеки, його використання забезпечить надійний захист паролів, мінімізуючи ризик їх викрадення навіть у разі фізичного доступу до пристрою, що

робить цю технологію особливо актуальною для користувачів, які прагнуть забезпечити максимальний рівень захисту своїх даних у цифровому середовищі.

1.2 Огляд існуючих програмних засобів для керування паролями

Керування паролями є критично важливим завданням у забезпеченні інформаційної безпеки. З огляду на зростаючу кількість онлайн-сервісів, користувачі змушені запам'ятовувати та використовувати численні паролі. Для полегшення цього завдання були створені програмні менеджери паролів, які дозволяють зберігати паролі у зашифрованому вигляді та автоматизувати процес їх введення, кожен з яких має певні переваги та недоліки.

Одним з найпопулярніших програмних засобів для керування паролями є Bitwarden – це менеджер паролів з відкритим вихідним кодом, який допомагає користувачам зберігати, управляти і захищати свої паролі та іншу конфіденційну інформацію. Програма доступна на різних платформах, включаючи Windows, macOS, Linux, Android, iOS, а також як розширення для основних веб-браузерів (Chrome, Firefox, Safari, Edge та інші) [6]. Bitwarden також пропонує плани для команд та підприємств, що дозволяють спільно використовувати паролі та інші дані в організаціях. На вибір доступні безкоштовний план, що включає основні функції для індивідуального використання, преміум план, який за невелику щорічну плату надає додаткові функції, такі як розширена підтримка двофакторної автентифікації, 1 ГБ захищеного зберігання файлів, пріоритетна підтримка та інші, а також сімейний план, що дозволяє ділитися обліковими записами і збереженими даними до 6 користувачів.

Іншим рішенням є LastPass, що також є досить популярним менеджером паролів, який дозволяє користувачам зберігати всі свої паролі в одному зашифрованому сховищі. Він забезпечує автоматичне заповнення форм для входу на веб-сайти, генерацію сильних паролів і синхронізацію даних між різними пристроями. Його основними функціями є автоматичне збереження та заповнення паролів, генератор паролів, зберігання нотаток та інших конфіденційних даних,

підтримка двофакторної автентифікації (2FA), та доступ з різних платформ, таких як браузери, мобільні додатки, десктопні програми [7].

1Password – це популярний менеджер паролів, відомий своєю зручністю та високим рівнем безпеки. Він підтримує зберігання паролів, конфіденційних нотаток та інших важливих даних, а також пропонує синхронізацію між пристроями через хмарне сховище. Він дозволяє окрім паролів до облікових записів зберігати дані кредитних карток та конфіденційних нотаток, підтримує інтеграцію зі сторонніми сервісами, є кросплатформеним, має високий рівень шифрування [8].

Ще одним рішенням, яке дещо відрізняється від перелічених вище своїм способом зберігання паролів є KeePass – це безкоштовний, відкритий менеджер паролів, який дозволяє користувачам зберігати паролі локально на своїх пристроях. Він забезпечує високий рівень захисту та гнучкі налаштування для користувачів, які хочуть контролювати свої дані [9]. Основний його функціонал багато в чому подібний до попередніх програмних засобів: KeePass дозволяє генерувати паролі, підтримує автозаповнення, двохфакторну автентифікацію, тощо. Поміж тим, даний засіб містить підтримку плагінів для розширення функціоналу. Може бути встановленим на флеш-носій, а також є повністю безкоштовним

Для порівняння наведених програмних засобів для керування паролями наведено таблицю (табл. 1.1), яка відображає їхні основні характеристики.

Огляд переваг та недоліків розглянутих програмних засобів дозволяє зробити такі висновки. LastPass можна порекомендувати для користувачів, які шукають зручний і легкий у використанні менеджер паролів з доступом на різних платформах, та бажають мати безкоштовну версію з основними функціями. Проте, Менеджер паролів LastPass зазнав кількох інцидентів з безпекою, які мали певний вплив на його репутацію, наприклад, у червні 2015 року LastPass повідомив про витік даних, який стався внаслідок несанкціонованого доступу до серверів [10].

Зловмисники отримали доступ до електронних адрес користувачів, гешованих паролів майстер-акаунтів, нагадувань паролів і деякої іншої інформації.

Таблиця 1.1 – Порівняльна таблиця засобів керування паролями

Характеристика	LastPass	1Password	Bitwarden	KeePass
Платформа	Web, Mobile, Desktop	Web, Mobile, Desktop	Web, Mobile, Desktop	Desktop, Mobile
Безкоштовна версія	Так	Ні	Так	Так
Шифрування	AES-256	AES-256	AES-256	AES-256
Двофакторна автентифікація	Так	Так	Так	Так
Синхронізація	Так	Так	Так	Обмежена
Характеристика	LastPass	1Password	Bitwarden	KeePass
Автозаповнення	Так	Так	Так	Так
Багатокористувацький доступ	Так	Так	Так	Ні
Відкритий код	Ні	Ні	Так	Так
Ціна	Від $3/місяць	Від $2.99/місяць	Безкоштовно/Від $1/місяць	Безкоштовно

1Password підходить користувачам, які готові платити за високий рівень безпеки та інтеграцію з іншими сервісами, а також тим, хто працює на різних платформах, однак в даного засобу повністю відсутня безкоштовна версія. Bitwarden підходить для тих, хто віддає перевагу відкритому вихідному коду і шукає бюджетне рішення з високим рівнем безпеки. KeePass є непоганим рішенням для користувачів, які хочуть повністю контролювати свої дані локально і не потребують автоматичної синхронізації між пристроями. Програмний засіб є повністю безкоштовним, але може бути дещо складним в освоєнні для недосвідченого користувача, та при недостатній захищеності персонального комп’ютера, або ж ноутбука на якому локально зберігаються паролі, може призвести до витоку даних.

На основі аналізу існуючих рішень можна зробити висновок, що в цілому досліджені програмні засоби для управління паролями мають в своїй основі схожий функціонал, такий як шифрування паролів при збереженні, генерація нових паролів, автозаповнення, тощо. Вибір конкретного засобу залежить від потреб користувача або підприємства, наявності багатокористувацького доступу,

кросплатформеності застосунку, ціни використання та потреби в наявності додаткового функціоналу, такого як захищене збереження та обмін файлів та інших функцій що можуть надаватися цими застосунками.

Аналіз існуючих програмних засобів для керування паролями показує, що кожен з розглянутих менеджерів має свої унікальні переваги та недоліки. Вибір конкретного рішення залежить від індивідуальних потреб користувача, таких як зручність використання, рівень безпеки, доступність на різних платформах та вартість. Незважаючи на численні переваги, усі програмні менеджери паролів мають певні обмеження щодо безпеки, зокрема, можливість компрометації даних через вразливості ПЗ або атаки зловмисників.

Ще однією спільною рисою для усіх перелічених менеджерів паролів є також той факт, що кожен з них потребує використання майстер-паролю, тобто паролю, який використовується для входу в обліковий запис конкретного менеджера, в якому зберігаються дані для автентифікації для усіх інших наявних у користувача акаунтів. При неналежному зберіганні такого майстер-паролю, або ж у разі хакерської атаки, зловмисник може потенційно отримати доступ до паролів від усіх облікових записів користувача, що є критичною проблемою інформаційної безпеки.

Окрім того, хоч користувачу потрібно пам’ятати тільки один пароль, проте він має бути надійним та складним, що може викликати певні незручності у використанні програми для доступу до записів що зберігаються в менеджері.

Це обґрунтовує актуальність розробки менеджера паролів з автентифікацією на основі апаратного токену, який може забезпечити додатковий рівень захисту та мінімізувати ризики, пов'язані з використанням програмних рішень, а також значно спростити процес реєстрації та автентифікації, за рахунок відсутності потреби в запам’ятовувані будь-яких паролів, при цьому не тільки не знизивши, а й підвищивши рівень безпеки, адже для отримання доступу до акаунту, потрібно скористатися фізичним пристроєм, наприклад, USB флеш-носієм, що зловмисник не зможе зробити, не отримавши при цьому фізичного доступу до апаратного токену.

1.3 Технології та стандарти апаратних токенів

Апаратні токени є одним з найефективніших засобів забезпечення інформаційної безпеки. Вони використовуються для захисту конфіденційних даних, автентифікації користувачів та захисту від несанкціонованого доступу. Це фізичні пристрої, які використовуються для автентифікації користувачів і забезпечення безпеки доступу до інформаційних систем [11]. Вони генерують та зберігають криптографічні ключі, необхідні для автентифікації та шифрування. Ці ключі ніколи не покидають пристрій, що забезпечує високий рівень захисту від витоку інформації.

Процес автентифікації з використанням такого токена включає насамперед його підключення користувачем до комп'ютера або мобільного пристрою через USB, NFC або Bluetooth. Після цього користувачу може бути потрібно ввести ПІН-код або використовувати біометричні дані для підтвердження своєї особи, в залежності від апаратного токену, який використовується. Після чого токен підписує автентифікаційний запит криптографічним ключем, зберігаючи його в таємниці, а сервер перевіряє підпис, підтверджуючи автентичність користувача.

Найпоширенішими апаратними токенами є фізичні ключі, такі як YubiKey, Google Titan і OnlyKey. Вони забезпечують автентифікацію через підключення до USB-порту або використання бездротових технологій, таких як NFC чи Bluetooth. Іншим типом токенів є смарт-карти, що містять вбудовані мікропроцесори і використовуються для зберігання цифрових сертифікатів і ключів. Вони забезпечують високу безпеку і часто використовуються здебільшого в корпоративному та урядовому секторах. Ще одним типом токенів є токени з екраном. Вони оснащені невеликими екранами, які відображають одноразові паролі (OTP), що змінюються через певні інтервали часу або генеруються у відповідь на запит.

Існує декілька стандартів та протоколів що найчастіше використовуються в апаратних токенах, які варто розглянути. Одним із таких стандартів є FIDO U2F (Universal 2nd Factor). це відкритий стандарт для двофакторної автентифікації, розроблений FIDO Alliance [12]. Його основна мета – забезпечити додатковий

рівень захисту для онлайн-акаунтів, використовуючи апаратні токени. Для використання користувачеві потрібно просто підключити апаратний токен до USB-порту (або використовувати бездротове з'єднання) і натиснути кнопку на токені для підтвердження автентифікації. FIDO U2F використовує автентифікацію на основі домену, що забезпечує захист від фішингових атак. Токен підтверджує справжність веб-сайту перед відправленням підпису. Також FIDO U2F підтримується багатьма великими онлайн-сервісами, включаючи Google, Facebook, Dropbox і GitHub. Принцип роботи полягає в тому, що користувач реєструє токен на підтримуваному веб-сайті, який генерує пару ключів (публічний і приватний). При кожному вході на сайт користувач підключає токен, який генерує унікальний підпис на запит автентифікації, використовуючи приватний ключ. Сервер перевіряє підпис, використовуючи публічний ключ, і надає доступ у разі успішної перевірки.

FIDO2 – це наступне покоління протоколу FIDO, яке включає додаткові можливості для безпарольної автентифікації [13]. FIDO2 складається з двох основних компонентів: протоколу Client to Authenticator Protocol (CTAP) та WebAuthn API. Даний протокол забезпечує високий рівень безпеки, запобігаючи фішинговим атакам і атакам з використанням вкрадених паролів, підтримується багатьма браузерами та операційними системами, включаючи Google Chrome, Mozilla Firefox, Microsoft Edge і Windows 10, а також надає користувачам змогу автентифікуватися без використання паролів, що підвищує зручність і безпеку. Користувач реєструє апаратний токен на підтримуваному веб-сайті через WebAuthn API, який генерує пару ключів. При автентифікації токен генерує криптографічний підпис на запит автентифікації, використовуючи приватний ключ. Сервер перевіряє підпис, використовуючи публічний ключ, і надає доступ у разі успішної перевірки.

OTP (One-Time Password) – це протокол, що генерує одноразові паролі для автентифікації користувачів. OTP паролі можуть бути статичними (попередньо згенеровані списки) або динамічними (генеруються у момент автентифікації) [14]. Одноразові паролі знижують ризик компрометації, оскільки кожен пароль використовується тільки один раз. Їх генерація може відбуватися автоматично, що знижує потребу у запам'ятовуванні складних паролів. Протокол такого типу

використовується для двофакторної автентифікації, у банківських системах, корпоративних мережах і онлайн-сервісах. Принцип роботи полягає в тому, що апаратний токен або мобільний додаток генерує одноразовий пароль на основі секретного ключа та поточного часу (Time-based OTP) або лічильника (HMAC-based OTP). Користувач вводить одноразовий пароль під час автентифікації, а сервер перевіряє введений пароль, порівнюючи його з очікуваним значенням, згенерованим на основі того ж секретного ключа і параметрів.

Ще одним важливим стандартом є PIV (Personal Identity Verification) [15]. Він розроблений для забезпечення автентифікації, шифрування та цифрового підпису у корпоративних та урядових системах. Використовується в смарт-картах та апаратних токенах для забезпечення високого рівня безпеки. PIV відповідає вимогам багатьох урядових і корпоративних стандартів, включаючи FIPS 201. На PIV токен записані криптографічні ключі та сертифікати. Для автентифікації користувач підключає токен до пристрою і вводить ПІН-код. Далі токен підписує автентифікаційний запит або дешифрує дані, використовуючи приватний ключ, а сервер, або інша сторона перевіряє підпис або дешифровані дані, використовуючи відповідні публічні ключі.

Основними перевагами апаратних токенів є забезпечення високого рівня захисту, оскільки криптографічні ключі ніколи не покидають пристрій. Це знижує ризик компрометації даних через шкідливе ПЗ або фізичне викрадення. Апаратні токени використовують автентифікацію на основі домену, що робить їх несприйнятливими до фішингових атак. Вони працюють тільки з легітимними сайтами, запобігаючи передачі ключів на підроблені ресурси. Додатковими перевагами слугують їх автономність та мобільність, адже багато токенів працюють незалежно від операційної системи або встановленого програмного забезпечення, що знижує ризик вразливостей, пов’язаних із ПЗ, а їх компактні розміри дозволяють легко переносити їх, забезпечуючи безпечний доступ до захищених систем у будь-якому місці.

Варто також зазначити деякі недоліки пов’язані з використанням подібних пристроїв. Перш за все користувач повинен мати фізичний доступ до токена для виконання автентифікації. Його втрата або поломка може суттєво ускладнити

доступ до системи. Окрім того, Не всі апаратні токени сумісні з усіма системами або додатками. Це може вимагати додаткових налаштувань або використання кількох токенів для різних платформ. Наприклад, для користування одним і тим самим застосунком на персональному комп’ютері та на смартфоні потрібно мати два різних токени, що пов’язано з відмінністю фізичних портів на таких пристроях. Ще одним недоліком є вартість використання таких рішень: апаратні токени є значно дорожчими, ніж програмні рішення для автентифікації, що може бути проблемою для окремих користувачів або невеликих компаній.

Нижче наведена таблиця (табл. 1.2), яка відображає порівняння характеристик найпопулярніших на сучасному ринку апаратних токенів.

Таблиця 1.2 – Порівняльна таблиця сучасних апаратних токенів

Характеристика	Google Titan	OnlyKey	YubiKey
Виробник	Google	CryptoTrust	Yubico
Типи підключення	USB-A, USB-C, Bluetooth	USB-A, USB-C	USB-A, USB-C, NFC
Підтримувані протоколи	FIDO U2F, FIDO2	FIDO U2F, OTP, PGP	FIDO U2F, FIDO2, OTP, PIV
Основні функції	Двофакторна автентифікація, захист від фішингу	Зберігання паролів, двофакторна автентифікація	Двофакторна автентифікація, захист від фішингу, зберігання паролів, PIV
Характеристика	Google Titan	OnlyKey	YubiKey
Безпекові характеристики	Захист від фішингових атак, криптографічний захист	Захист паролів, криптографічний захист, автономна робота	Захист від фішингових атак, криптографічний захист, водонепроникність
Ціна	Відносно висока	Відносно висока	Широкий діапазон, залежно від моделі
Відсутність ПЗ	Так	Так	Так
Фізичні розміри	Компактні, портативні	Компактні, портативні	Компактні, портативні
Сумісність	Google, Windows, macOS, Linux	Windows, macOS, Linux	Google, Windows, macOS, Linux, iOS, Android
Додаткові функції	Захист облікових записів Google	Резервне копіювання паролів, підтримка PGP	Підтримка PGP, керування паролями, режим NFC
Надійність	Висока	Висока	Висока

Аналізуючи представлені в таблиці характеристики можна зробити певні висновки. Google Titan є надійним рішенням для забезпечення двофакторної автентифікації, розробленим для захисту облікових записів Google та інших онлайн-сервісів. Підтримує протоколи FIDO U2F і FIDO2, забезпечуючи захист від фішингових атак. Підключається через USB-A, USB-C або Bluetooth, що робить його зручним для використання на різних пристроях. OnlyKey від CryptoTrust пропонує багатофункціональне рішення для зберігання паролів і двофакторної автентифікації. Підтримує різноманітні протоколи, включаючи FIDO U2F, OTP та PGP, що робить його універсальним інструментом для безпеки. YubiKey від Yubico є одним з найпопулярніших апаратних токенів на ринку. Він підтримує широкий спектр протоколів, включаючи FIDO U2F, FIDO2, OTP і PIV, що дозволяє використовувати його в різних середовищах.

Таким чином, апаратні токени є надійним засобом для захисту даних і автентифікації користувачів, забезпечуючи високий рівень безпеки завдяки використанню криптографічних методів. Вони є незамінними в умовах, де потрібен високий рівень захисту інформації. Сучасні токени, такі як YubiKey, Google Titan і OnlyKey, демонструють ефективність і надійність у забезпеченні безпеки в цифровому світі.

1.4 Формалізація вимог та постановка задачі

Для успішної розробки програмного засобу керування паролями необхідно чітко визначити вимоги та завдання, які система має виконувати. Вимоги повинні враховувати всі аспекти функціональності, безпеки, продуктивності та зручності використання, щоб забезпечити високий рівень задоволення потреб користувачів та відповідність технічним стандартам. Цей підрозділ формалізує вимоги до системи та визначає конкретні завдання, що мають бути вирішені під час розробки.

Головною функцією даного засобу є забезпечення надійного зберігання та відображення паролів користувацьких облікових записів. Програма має підтримувати базові операції з керування даними: зчитування, зберігання, оновлення та видалення записів. Це включає роботу з обліковими записами

користувачів, їх логінами та паролями, що зберігаються в базі даних. Система повинна забезпечувати швидкий і надійний доступ до записів, а також можливість їх модифікації та видалення відповідно до потреб користувачів.

Важливою вимогою до програмного засобу є забезпечення безпеки збережених даних. Для цього логіни та паролі повинні шифруватися перед збереженням у базу даних і дешифруватися при їх отриманні. Використання надійних криптографічних алгоритмів гарантує, що дані залишаться захищеними навіть у разі несанкціонованого доступу до бази даних. Окрім того, повна інформація збережених записів, яка включає безпосередньо логін та пароль повинна отримуватися з бази даних і відображатися лише тоді, коли відповідний запит до серверної частини застосунку здійснено користувачем через взаємодію з веб-інтерфейсом. Такий підхід дозволить підвищити безпеку та зменшити кількість конфіденційної інформації що передаватиметься мережею.

Однією з ключових функцій засобу керування паролями та його особливістю є реєстрація та автентифікація користувачів за допомогою USB флеш-носія, що виконує роль апаратного токена. Це включає зчитування інформації про назву та серійний номер USB пристрою для ідентифікації користувача, після чого з отриманої інформації формується текстовий рядок що підлягає шифруванню та подальшому збереженні в базі даних, який буде являти собою майстер-пароль від облікового запису користувача в менеджері паролів. Такий метод дозволить швидко проходити процеси автентифікації та реєстрації, при цьому не вимагаючи запам’ятовування жодних паролів. Завдяки цьому підвищується рівень безпеки та зменшується ризик несанкціонованого доступу, адже для входу в обліковий запис потрібно мати фізичний доступ до носія. У разі ж втрати або поломки USB пристрою, має бути передбачена можливість скидання майстер-паролю та створення нового.

Для покращення організації даних користувачі повинні мати можливість створювати групи, які виконуватимуть роль папок для сортування записів. Це дозволить структурувати інформацію, полегшуючи доступ до необхідних записів та їх управління. Групи можуть бути створені на основі категорій, таких як типи облікових записів або рівні доступу.

Програмний засіб повинен надавати функцію генерації надійних паролів відповідно до параметрів, заданих користувачами. Це включає довжину пароля, використання великих та малих літер, цифр і спеціальних символів. Така функціональність забезпечує користувачам можливість створювати сильні паролі, що підвищують безпеку їх облікових записів.

Окрім того важливо реалізувати зручний та інтуїтивно зрозумілий інтерфейс. Він повинен бути простим у використанні, забезпечувати легкий доступ до всіх функцій програми та підтримувати користувачів у виконанні всіх операцій. Зручність інтерфейсу сприяє підвищенню ефективності роботи користувачів з програмою та покращенню загального користувацького досвіду.

Формалізація цих вимог та постановка відповідних задач є важливим етапом у розробці програмного засобу для керування паролями, що забезпечить його ефективну та безпечну роботу, а також задоволення потреб користувачів.

1.5 Висновки до розділу аналіз предметної області

У даному розділі було проведено детальний огляд існуючих програмних засобів для керування паролями. Це дозволило виявити основні переваги та недоліки сучасних рішень, що стало основою для визначення вимог до нового програмного засобу. Досліджено технології та стандарти апаратних токенів, такі як FIDO U2F, FIDO2, OTP (One-Time Password) та PIV (Personal Identity Verification).. Формалізовано вимоги та здійснено постановку задачі. На основі проведеного аналізу та досліджень було визначено ключові функціональні та нефункціональні вимоги, що забезпечили чітке розуміння необхідних характеристик та функцій програмного засобу.

2 ПРОЕКТУВАННЯ ЗАСОБУ

2.1 Розробка архітектури засобу

Архітектура програмного засобу для керування паролями складається з кількох основних модулів, які відповідають за виконання необхідних функцій та забезпечення ефективної взаємодії між компонентами системи. Кожен модуль має чітко визначені завдання і функції, що дозволяє оптимізувати роботу всього програмного забезпечення.

Загальна структура програмного засобу наведена на рисунку 2.1.

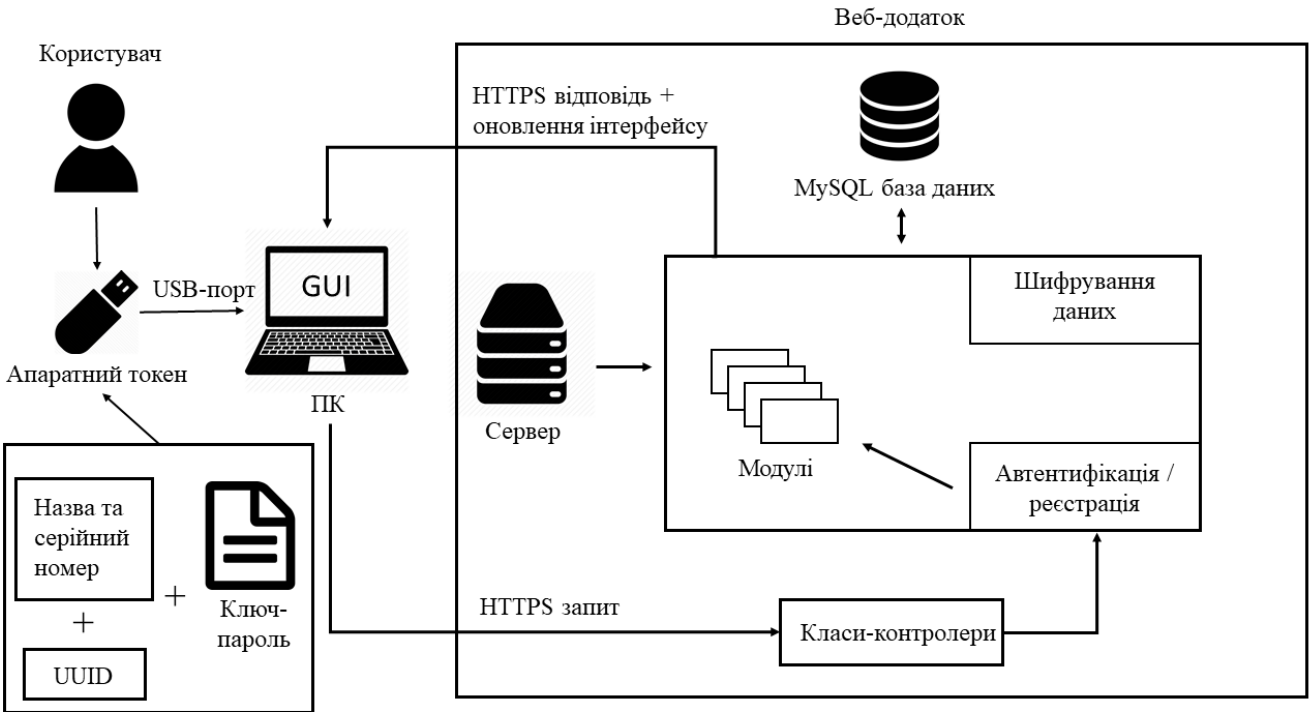


Рисунок 2.1 – Загальна структура програмного засобу

Користувач, приєднуючи апаратний токен до свого персонального комп’ютера або ж ноутбука для проходження процесів реєстрації та автентифікації, взаємодіє з програмним засобом через графічний інтерфейс, внаслідок чого, на сервер відправляються запити що обробляються класами-контролерами (англ. Controller) [16], після чого виконується певна програмна логіка, а за потреби відбувається взаємодія з базою даних. В результаті формується відповідь сервера на запит, яка повертається до клієнтської сторони, а також оновлюється інтерфейс.

Нижче схематично зображено основні модулі, що формують архітектуру програми (рис. 2.2):

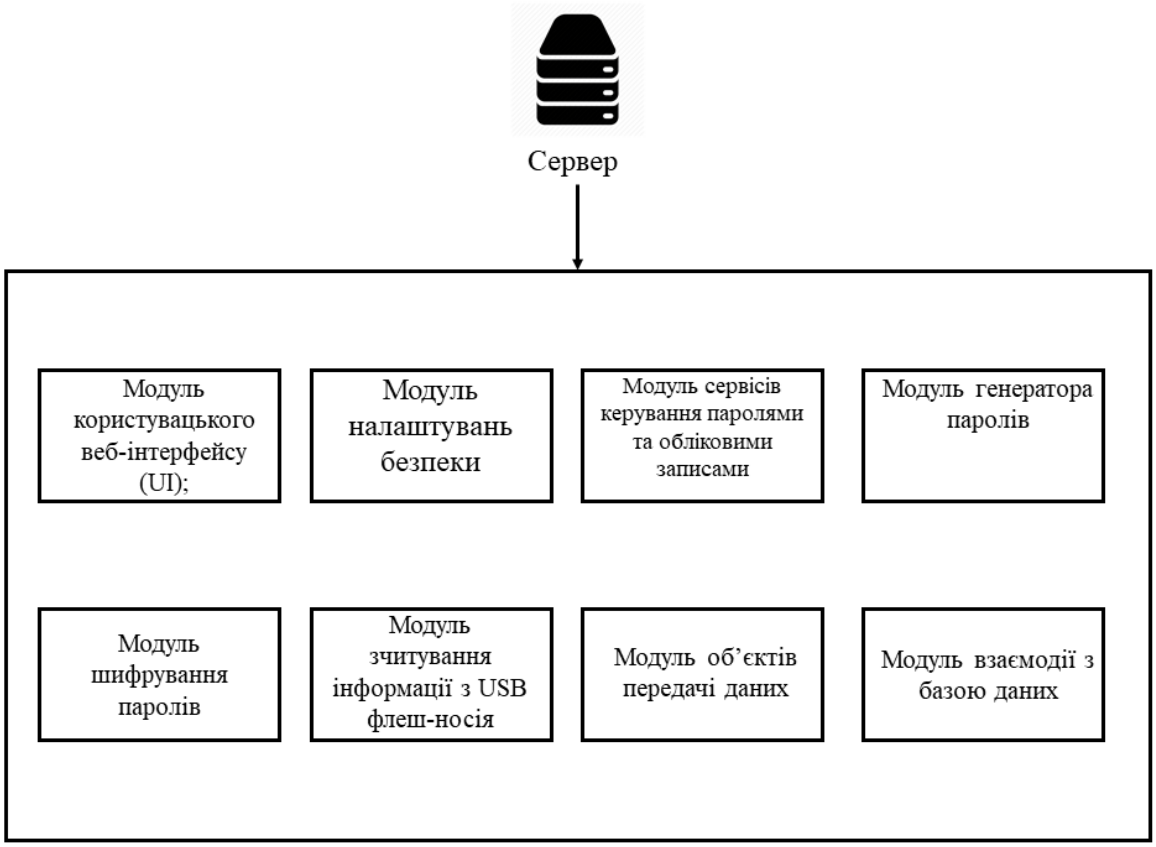


Рисунок 2.2 – Схематичне зображення модулів програмного засобу

Взаємодія користувача з програмним засобом відбувається за допомогою модуля користувацького веб-інтерфейсу, який надає змогу виконувати операції читання, збереження, оновлення та видалення записів що створює користувач, здійснювати їх пошук та сортування, генерувати надійні паролі, а також керувати обліковим записом користувача в застосунку. Усі дії що відбуваються на клієнтській стороні при використанні графічного інтерфейсу обробляються класами-контролерами, які взаємодіють з відповідними сервісами керування паролями та обліковими записами, що в свою чергу виконують певну бізнес-логіку та звертаються до модуля взаємодії з базою даних, який зберігає усю необхідну інформацію. Модуль налаштувань безпеки визначає правила створення сесії між клієнтською та серверною частиною програми, а також правила доступу до різних елементів інтерфейсу [17]. Модуль генератора паролів генерує стійкі паролі, та дозволяє задавати параметри генерації, після чого отриманий пароль можна

зберегти до бази даних створивши запис. Модуль шифрування паролів відповідає за шифрування та дешифрування потрібної інформації з бази даних для її безпечного зберігання та відображення на вимогу користувача. Модуль зчитування інформації з USB флеш-носія спрацьовує при спробі реєстрації або автентифікації в системі та дозволяє отримувати інформацію про носій, яка буде зашифрована модулем шифрування і використовуватиметься в якості майстер-паролю до облікового запису. Модуль об’єктів передачі даних дає змогу відображати лише необхідну в конкретний момент інформацію, замість усієї що повертається з бази даних.

Для кращого розуміння взаємодії компонентів системи далі буде наведено більш детальний опис кожного з модулів.

Модуль користувацького веб-інтерфейсу надає користувачеві взаємодіяти з програмним засобом. Для створення інтерфейсу було використано фреймворк «Vaadin», що дозволяє легко та швидко створювати зручні, функціональні та прості у користуванні веб-інтерфейси. Користувач має змогу створити обліковий запис, пройти автентифікацію, створювати записи, що містять усю необхідну інформацію, а саме логін та пароль від облікового запису, назву запису, опис, посилання на сторінку входу в обліковий запис. Також є можливість генерування паролів, вказавши необхідні параметри, такі як довжина та символи які мають бути задіяні та створення груп, які виконують роль папок, до яких можна додавати наявні записи для сортування за категоріями та зручного доступу до них. Увесь інтерфейс при використанні фреймворку «Vaadin» складається з компонентів, що створюються за допомогою програмного коду в класах-контролерах та мають прослуховувачі подій. Таким чином, наприклад, при натисканні кнопки збереження запису викликається потрібний метод сервісу, що приймає дані введені користувачем та зберігає їх до бази даних.

Модуль налаштувань безпеки відповідає за конфігурацію та управління аспектами безпеки в додатку. Він дозволяє визначити правила доступу до різних ресурсів, налаштувати автентифікацію та авторизацію користувачів, а також інтеграцію з різними системами автентифікації. Цей модуль містить налаштування для забезпечення захисту URL-адрес, визначаючи, які запити потребують

автентифікації, а які доступні публічно. Крім того, у цьому модулі налаштовано політики сеансів, управління користувачами, а також обробку виключень і помилок доступу.

Модуль сервісів керування паролями та обліковими записами є одним з основних в програмному засобі, адже він, в свою чергу, взаємодіє з модулями шифрування паролів, модулем взаємодії з базою даних, а також об’єктами передачі даних, забезпечуючи виконання усіх необхідних функцій програмного засобу.

Фундаментом архітектури засобу керування паролями став MVC (Model-View-Controller) – це архітектурний шаблон, який використовується для створення програмного забезпечення з розділенням функціональності на три основні компоненти: модель, представлення та контролер. Цей підхід дозволяє забезпечити чітке розділення обов'язків, що полегшує розробку, тестування та підтримку програм.

Модель описує дані та бізнес-логіку програми. Вона відповідає за управління даними, їх обробку та взаємодію з базою даних або іншими джерелами даних. Модель не знає про існування представлення чи контролера, що забезпечує її незалежність та можливість повторного використання. Сюди відносяться модулі взаємодії з базою даних, об’єктів передачі даних, зчитування інформації з USB флеш-носія, шифрування паролів, генератора паролів, а також сервіси керування паролями та обліковими записами.

Представлення (або View) відповідає за відображення даних користувачеві. Воно отримує дані від моделі через контролер і відображає їх у зручному для користувача вигляді. Представлення не містить бізнес-логіки, а лише відповідає за візуальне подання інформації.

Контролер (або Controller) виступає посередником між моделлю та представленням. Він обробляє вхідні запити від користувача, викликає відповідні методи моделі для обробки даних, а потім передає результати представленню. Контролер забезпечує, що дані від моделі правильно відображаються в представленні, та обробляє дії користувача, такі як натискання кнопок або введення даних у форми. Так як при розробці засобу використовувався фреймворк Vaadin, модуль користувацького веб-інтерфейсу можна віднести одразу до рівня і

представлення, і контролера, адже класи-контролери в даному випадку відповідають не лише за обробку запитів що надходять від користувача внаслідок взаємодії з інтерфейсом, але і за його створення.

MVC архітектура працює таким чином: коли користувач взаємодіє з інтерфейсом користувача (представленням), контролер отримує цей запит і вирішує, які дії потрібно виконати. Контролер взаємодіє з моделлю для обробки даних або виконання бізнес-логіки, після чого оновлює представлення з новими або зміненими даними. Такий підхід до розробки дозволяє чітко розділити відповідальність між різними частинами програми, що робить код більш організованим і зрозумілим. Розділення на модель, представлення та контролер сприяє покращенню підтримуваності та масштабованості системи, оскільки зміни в одному компоненті мінімально впливають на інші. MVC також полегшує тестування, оскільки кожен компонент можна тестувати окремо. Завдяки цим перевагам, MVC є одним з найпопулярніших архітектурних шаблонів для розробки веб-додатків та інших програмних систем.

2.2 Розробка алгоритмів функціонування програми

Алгоритми роботи засобу керування паролями є ключовим елементом його ефективності. У даному підрозділі буде описано алгоритми, що відповідають за зчитування інформації про USB флеш-носій, реєстрацію та автентифікацію користувачів, виконання різних операцій із записами які містять інформацію про користувацькі акаунти, взаємодію з базою даних, генерацію та шифрування паролів, а також відображення даних в інтерфейсі.

Алгоритм реєстрації користувача починається з отримання інформації введеної у форму реєстрації. Після заповнення даних, необхідно під’єднати USB-носій до персонального комп’ютера або ноутбука та натиснути кнопку «Зареєструватися». Далі запускається алгоритм зчитування інформації з носія, що використовує спеціально створену .dll-бібліотеку яка отримує потрібну інформацію про флеш-носій, а саме його назву та серійний номер, і передає в метод сервісу. Далі відбувається отримання літери логічного диску що пов’язаний з

носієм, унікального UUID та генерація надійного паролю довжиною в сто символів, який записується у файл і зберігається на USB-пристрої за рахунок раніше визначеного логічного диску. Майстер-пароль від облікового запису в менеджері паролів формується шляхом комбінування інформації про носій та згенерованого паролю який було збережено у файл. Було розглянуто можливість збереження даного паролю в альтернативний потік NTFS [18, 19], однак такий підхід не є виправданим в даній реалізації, оскільки такий тип файлової системи не використовується в більшості USB-носіях. Після чого буде спочатку створено нового користувача в базі даних, попередньо зашифрувавши електронну пошту та майстер-пароль, а потім відбудеться процес автентифікації, що дозволить отримати щойно зареєстрованому користувачеві доступ до функціоналу менеджера паролів. Схему алгоритму зчитування інформації з USB-носія наведено на рисунку 2.3.

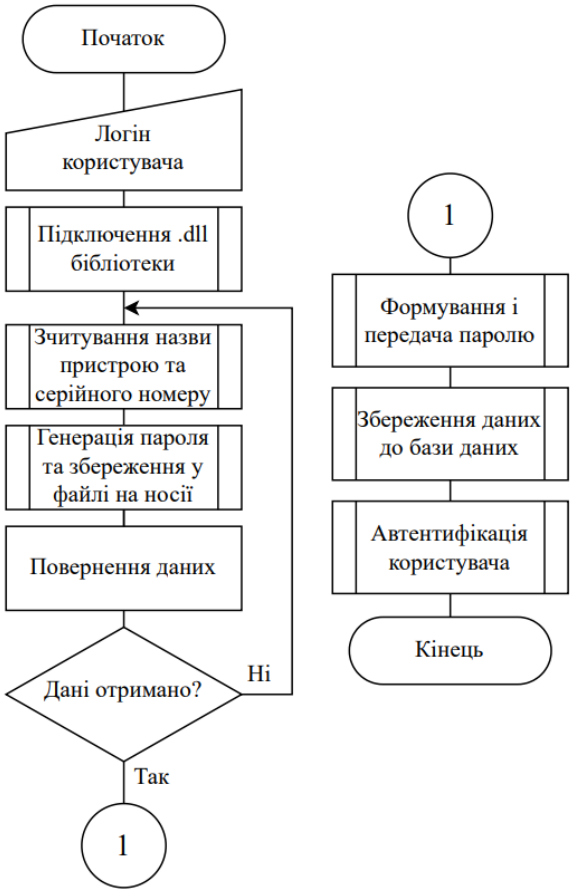


Рисунок 2.3 – Схема роботи алгоритму зчитування інформації з апаратного токену

Алгоритм автентифікації користувача відбувається схожим чином з алгоритмом реєстрації, за вийнятком процесу створення нового запису в базі даних. Після введення логіну, запускається алгоритм зчитування інформації з апаратного токена, зокрема зчитується вміст файлу який було згенеровано при реєстрації. отримані дані з носія записуються у форматі текстового рядка, який шифрується. Якщо користувач з вказаним логіном існує і при цьому сформований зашифрований рядок з інформацією про токен співпадає з полем майстер-пароля в базі даних, автентифікація пройде успішно, а користувач отримає доступ до функціоналу застосунку. Загальну схему алгоритму наведено на рисунку 2.4.

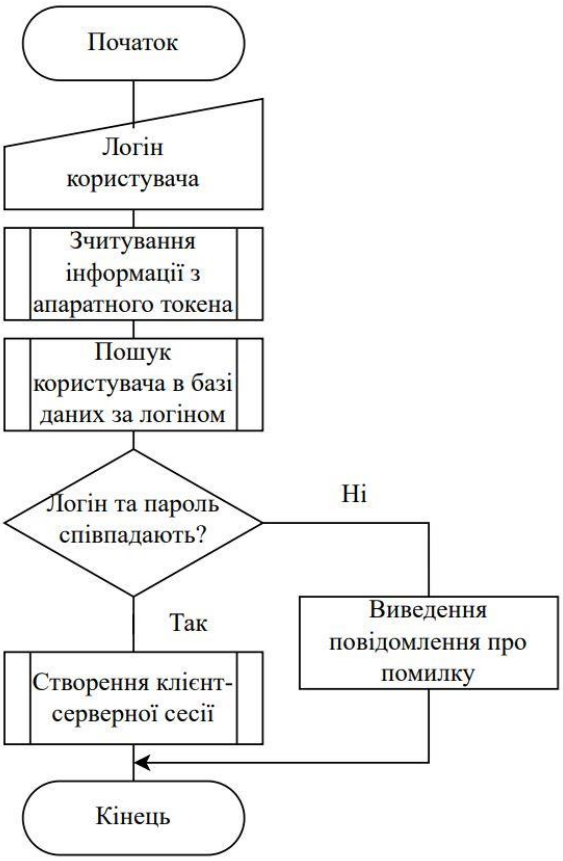


Рисунок 2.4 – Схема роботи алгоритму автентифікації користувача

Робота алгоритму виконання операцій читання, збереження, оновлення та видалення записів починається з додавання прослуховувачів подій на функціональні кнопки веб-інтерфейсу. При обробці події взаємодії з такими компонентами буде відправлено http-запити, які опрацьовують контролери, що викликають відповідні методи сервісу керування паролями, які в свою чергу

викноують певну бізнес-логіку та передають оброблені дані до одного з репозиторіїв, що є частиною модуля взаємодії з базою даних, який забезпечує виконання потрібних для конкретного типу операції (читання, оновлення, тощо) SQL запитів [20]. Нижче зображено візуальну схему передачі запитів для виконання операцій в базі даних із записами які створює користувач, та групами в які можна додавати записи для розділення їх за категоріями (рис. 2.5).

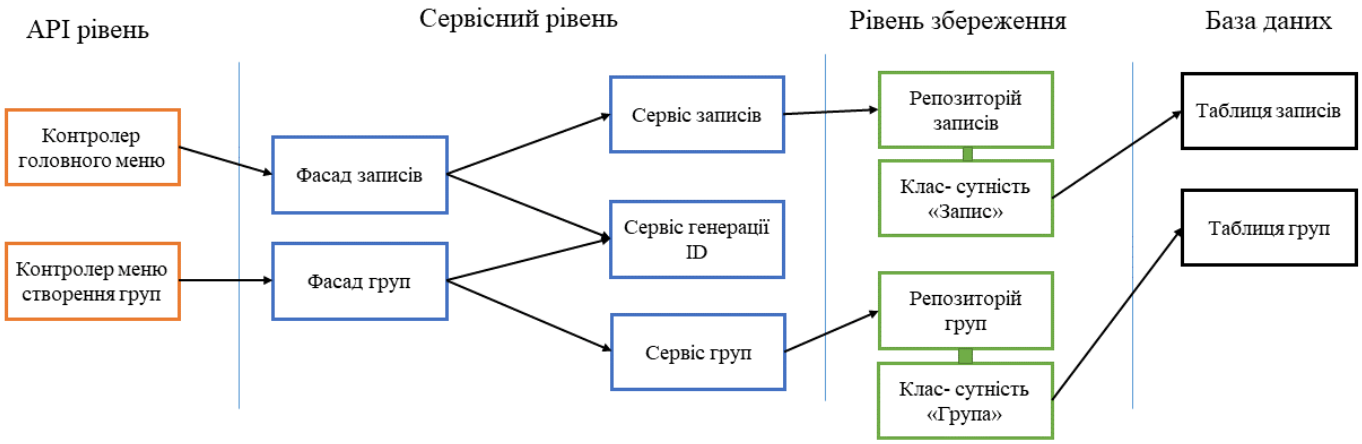


Рисунок 2.5 – Схема передачі запитів на виконання операцій читання, збереження, оновлення та видалення записів до бази даних

Контролери обробляють отриманий запит, викликаючи класи-фасади. Патерн проектування «Фасад» надає єдиний інтерфейс для взаємодії з комплексною системою, спрощуючи використання її складних компонентів [21]. Його основна мета - заховати складність системи, забезпечуючи більш зручний і зрозумілий інтерфейс для користувача. Фасад об'єднує підсистеми, які можуть мати численні класи і складні взаємодії, і представляє їх як один об'єкт, з яким легко працювати. Використання фасаду дозволяє зменшити залежність клієнтського коду від деталей реалізації підсистем, що полегшує підтримку та модифікацію коду. Замість того, щоб взаємодіяти з численними об'єктами безпосередньо, клієнт звертається до фасаду, який виконує всі необхідні операції, викликаючи відповідні методи внутрішніх об'єктів. Це не тільки спрощує розробку, але й покращує структуру програми, роблячи її більш зрозумілою та організованою. Класи-фасади звертаються до методів сервісів з якими вони працюють, які виконують певну бізнес-логіку, після чого викликаються методи репозиторіїв, які в свою чергу

формують запити до бази даних за допомогою Hibernate Query Langugage (HQL) що забезпечує виконання необхідних операцій з даними [22].

Генерація паролів починається з отримання даних з відповідного інтерфейсу налаштування параметрів генерації. Після вказання користувачем необхідної довжини пароля, символів що мають бути використані та натискання кнопки генерації, викликається сервіс, один з методів якого приймає потрібні вхідні дані та на їх основі формує текстовий рядок і повертає його до контролеру, який в свою чергу оновлює інтерфейс, помістивши отримані з методу дані до текстового поля, звідки згенерований пароль може бути скопійований та вставлений у форму створення нового запису, після чого відбудеться шифрування та збереження інформації в базі даних.

Алгоритм генерації паролів зображено на рисунку 2.6.

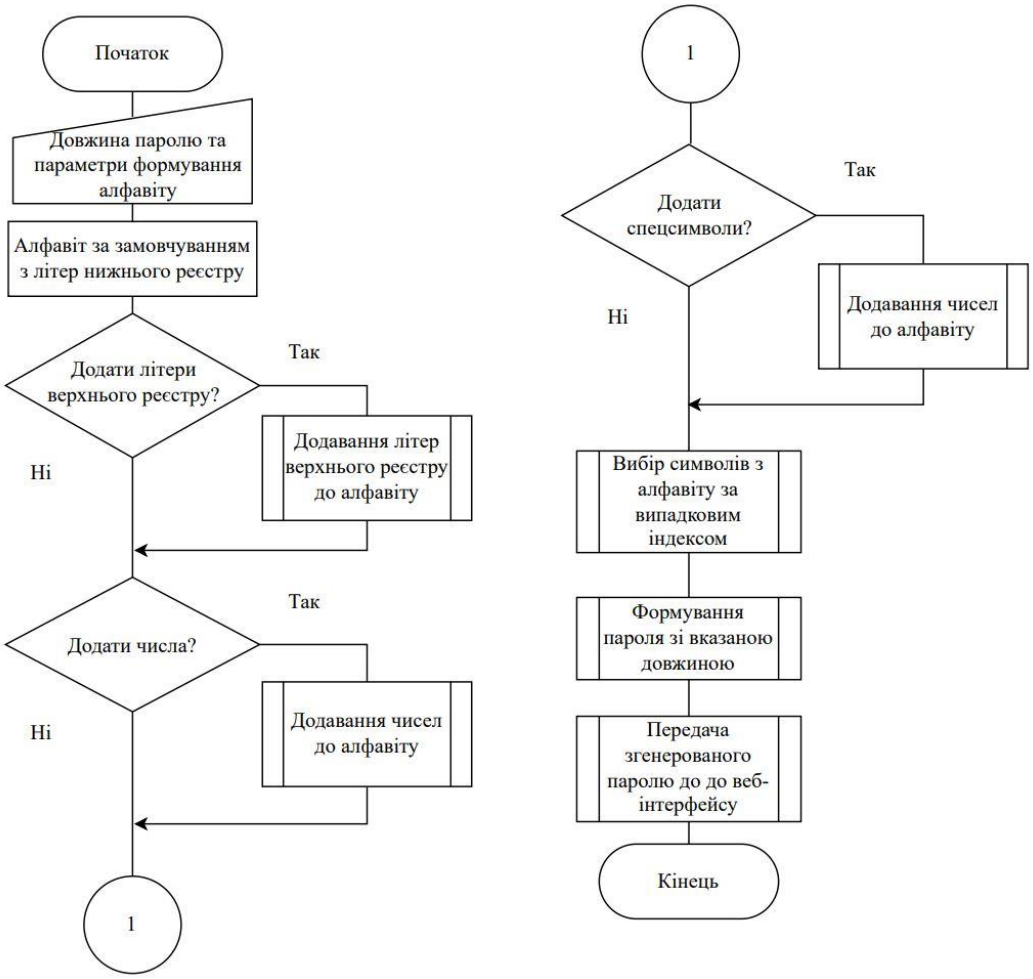


Рисунок 2.6 – Схема роботи алгоритму генерації паролів

Відображення інформації отриманої з бази даних в графічному інтерфейсі відбувається за допомогою спеціальних об’єктів для передачі даних, які називаються DTO (Data Transfer Object) [23]. Це паттерн проектування, який використовується для перенесення даних між різними шарами або компонентами системи. Основна суть DTO полягає в тому, що вони є простими об'єктами, які не містять бізнес-логіки, а лише дані, які необхідно передати. DTO часто використовуються для передачі даних між сервером та клієнтом, між різними модулями або сервісами в межах одного додатку. Вони дозволяють об'єднувати дані з кількох об'єктів або таблиць в один об'єкт, що зменшує кількість запитів до бази даних або інших джерел даних. Це підвищує ефективність передачі даних та зменшує навантаження на мережу.

Схему роботи модуля наведено на рисунку 2.7.

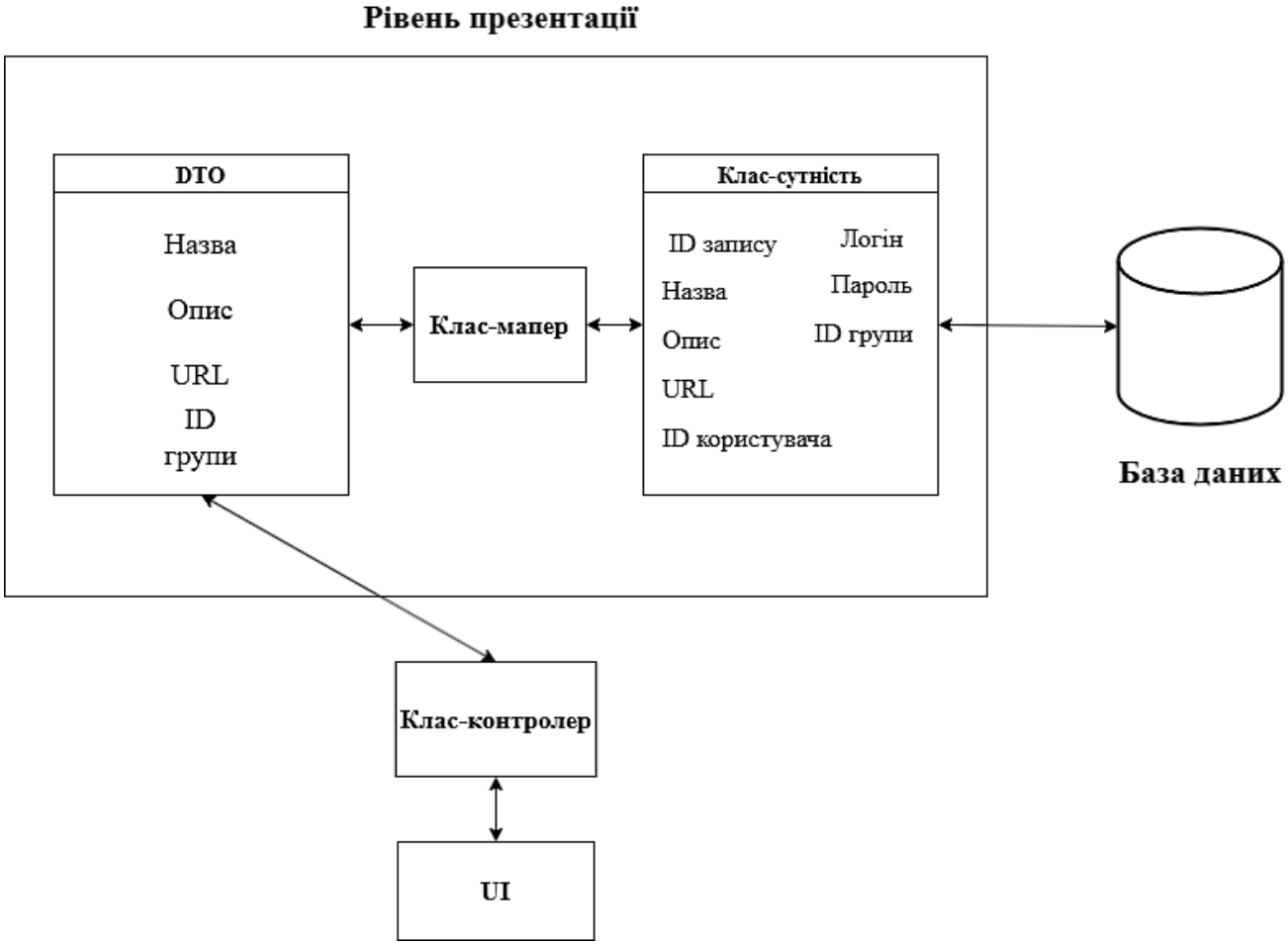


Рисунок 2.7 – Схема роботи модуля відображення інформації отриманої з бази даних

Використовуючи DTO, можна чітко розділити бізнес-логіку та дані, що відображаються користувачу. Це підвищує читабельність та підтримуваність коду, оскільки кожен шар виконує свою конкретну функцію. Також ці об’єкти дозволяють передавати лише необхідні дані, приховуючи внутрішні деталі реалізації та структуру бази даних, що зменшує ризик витоку чутливих даних та підвищує безпеку системи.

Mapper-класи, що заповнюють поля DTO, використовуються для переносу даних між різними шарами додатка [24]. Вони допомагають зменшити зв'язність між моделями бізнес-логіки та представленням даних, забезпечуючи структуроване та зрозуміле передавання даних. Основна задача таких класів – перетворювати об'єкти одного типу в об'єкти іншого типу. Наприклад, це може бути конвертація об'єктів сутностей (Entity) з бази даних у DTO для передачі на фронтенд. Такі мапери часто використовуються у багатошаровій архітектурі додатків для спрощення процесу перенесення даних між рівнями. Mapper-класи можуть бути реалізовані вручну або з використанням бібліотек автоматичного мапінгу. В ручних реалізаціях методи таких класів явно вказують, як кожне поле одного об'єкта слід перенести в поле іншого. Автоматичні mapper-класи, з іншого боку, надають більш декларативний підхід, зменшуючи кількість коду, який потрібно писати вручну.

Для шифрування майстер-пароля перед збереженням в базу даних було використано вбудовані можливості Spring Boot фреймфорку, а саме клас BCryptPasswordEncoder [25], що є частиною модуля Spring Security. Він використовує алгоритм BCrypt для шифрування паролів. Перед тим як зашифрувати пароль, BCryptPasswordEncoder генерує випадкову "сіль" (salt). Сіль додається до пароля, щоб зробити його більш захищеним від атак типу "rainbow table". Після додавання солі пароль хешується за допомогою алгоритму BCrypt. Зашифрований пароль разом із сіллю зберігається в базі даних. Це виглядає як рядок, що містить як зашифрований пароль, так і сіль. BCrypt використовує Blowfish у режимі CBC (Cipher Block Chaining) для створення стійкого до атак хеша паролів [26].

Для шифрування логінів та паролів які користувач зберігає у програмному засобі було обрано алгоритм шифрування AES (Advanced Encryption Standard) [27]. Даний алгоритм є одним із найпоширеніших і найбезпечніших сучасних симетричних блочних шифрів. Він був затверджений Національним інститутом стандартів і технологій США (NIST) у 2001 році як стандарт для шифрування електронних даних. Було використано вбудовану в Java бібліотеку javax.crypto, що надає реалізацію даного алгоритму. AES забезпечує високий рівень криптографічного захисту, який наразі не був зламаний при використанні правильних параметрів і налаштувань. Також він є міжнародним стандартом і підтримується практично всіма сучасними програмними і апаратними засобами шифрування. Багато сучасних процесорів мають вбудовану підтримку AES, що дозволяє виконувати шифрування та дешифрування дуже швидко і з низьким споживанням ресурсів.

Для того, щоб реалізувати шифрування за допомогою даного алгоритму має бути згенеровано вектор ініціалізації та ключ шифрування. Для генерації ключа було створено власний алгоритм, що бере за основу майстер-пароль користувача. Спочатку визначається середина рядка, після чого він розділяється навпіл. Потім для кожної з половин викликається метод, який переставляє сусідні символи місцями. Наприклад, для рядка "abcd" вихід буде "badc". Після перестановки символів у кожній половині вони об'єднуються в новий рядок, причому друга половина додається попереду першої. Якщо довжина отриманого рядка перевищує 32 символи, він обрізається до перших 32 символів. Нарешті, отриманий рядок конвертується у масив байтів, з якого створюється секретний ключ типу AES (рис. 2.8). Таким чином, отриманий ключ буде унікальним для кожного користувача, так як він створюється на основі майстер-паролю. Це означає, що за допомогою такого ключа можна розшифрувати лише ті дані, які безпосередньо належать відповідному користувачу, а не одразу усі дані з бази даних. Такий підхід забезпечить додатковий рівень безпеки даних, що зберігаються.

У разі втрати апаратного токена, або його фізичної несправності, користувач має змогу здійснити оновлення майстер-пароля, шляхом прив’язки до нового пристрою, при цьому не втративши доступ до збережених в базі даних записів.

Окрім того, необхідно забезпечити цілісність таких даних як логіни та паролі, адже ця інформація зберігається в зашифрованому вигляді, а ключ шифрування формується на основі майстер-пароля, який підлягає оновленню.

Рисунок 2.8 – Схема алгоритму генерації секретного ключа шифрування

Схему роботи даного алгоритму наведено на рисунку 2.9.

Алгоритм починається з перевірки електронної пошти. Спочатку користувач вводить свою електронну адресу. Далі перевіряється, чи зареєстрована ця пошта в системі. Якщо пошта не зареєстрована, виводиться повідомлення про помилку, і процес повертається на початок. Якщо пошта зареєстрована, на неї відправляється код верифікації. Після цього користувач повинен ввести отриманий код верифікації. Коли код введено, алгоритм переходить до наступного етапу, де перевіряється його правильність. Якщо код верифікації невірний, виводиться

повідомлення про помилку, тимчасовий змінний код очищується, і користувач перенаправляється на початкову URL-адресу.

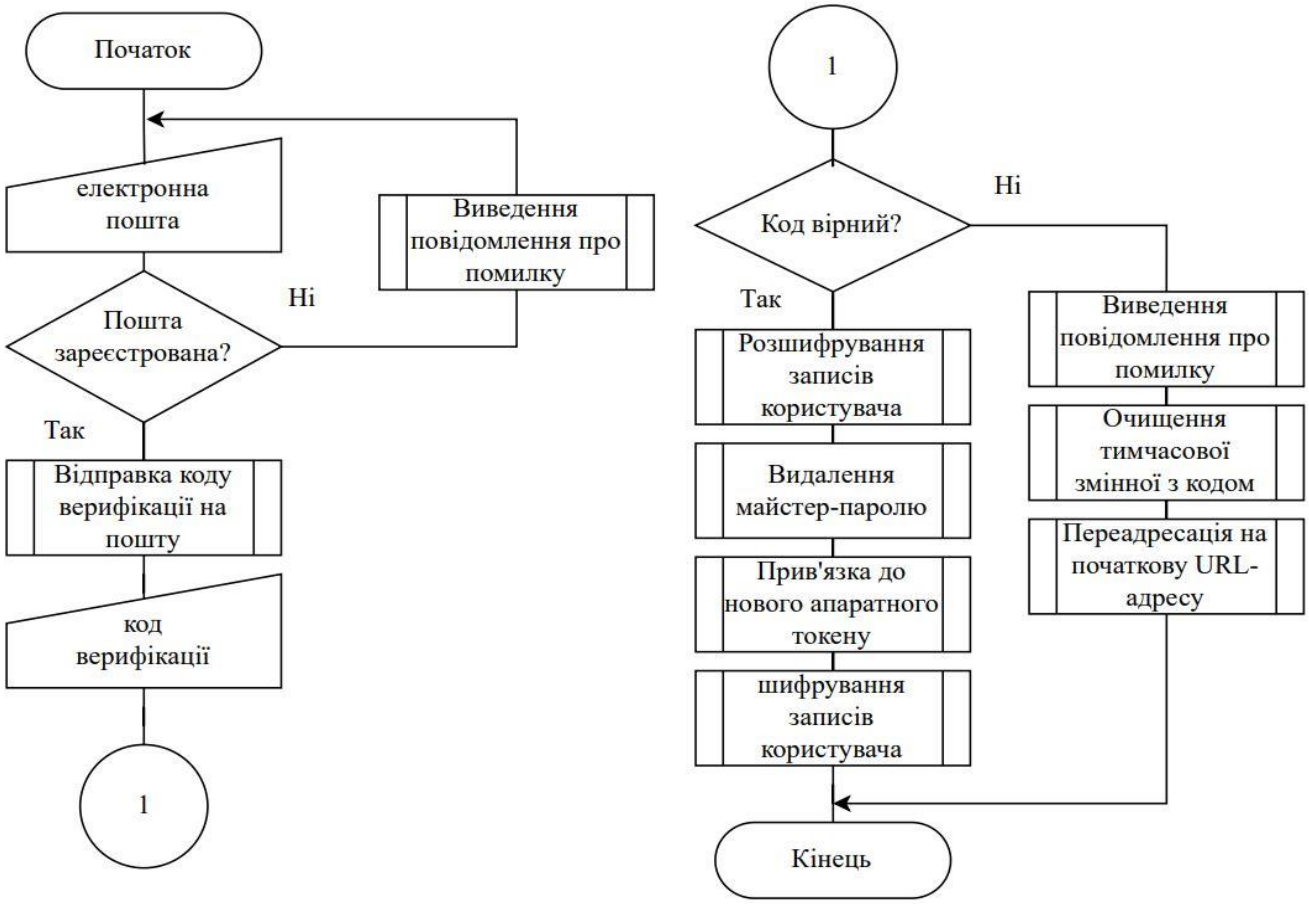


Рисунок 2.9 – Схема роботи алгоритму оновлення майстер-паролю та прив’язки до нового апаратного токена

Якщо код правильний, розшифровуються записи користувача. Після цього видаляється майстер-пароль, і виконується прив'язка до нового апаратного токена. Записи користувача знову шифруються. На цьому етапі алгоритм завершується.

2.3 Висновки до розділу проектування системи

Отже, в даному розділі виконано проектування системи, розроблено її загальну структуру, наведено основні компоненти, а також описано основні алгоритми програмного засобу, які відповідають за виконання найважливіших функцій, а саме прив’язки до апаратного токену, реєстрації та автентифікації користувачів, шифрування даних, тощо. Розроблені та схематично зображені принцип роботи даних алгоритмів. На основі розроблених рішень в наступному розділі розроблено програмний засіб.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ

3.1 Обґрунтування вибору інструментальних засобів розробки

В даному підрозділі розглядається ключовий аспект процесу розробки програмного засобу для керування паролями – вибір інструментальних засобів, які визначають технологічну основу проекту. Ефективний вибір інструментів розробки є важливою передумовою успішної реалізації програмного засобу та впливає на його якість, продуктивність та зручність у використанні. Буде зроблено акцент на обґрунтуванні вибору конкретних інструментів розробки, таких як мови програмування, фреймворки, середовища розробки та інші. Враховуючи специфіку завдань програмного забезпечення для керування паролями, необхідно обрати інструменти, що найкращим чином відповідають вимогам до безпеки, функціональності та продуктивності. Детальний аналіз різноманітних варіантів інструментів розробки дозволить визначити оптимальний набір технологій для реалізації програмного засобу керування паролями, забезпечуючи при цьому якісний та ефективний результат.

Для розробки веб-додатку обрано середовище розробки IntelliJ IDEA Ultimate Edition [28]. Це інтегроване середовище розробки, яке відоме своєю потужністю та функціональністю для роботи з мовами програмування Java, Kotlin, іншими JVM-мовами та веб-технологіями. Однією з основних переваг IntelliJ IDEA є його високий рівень автоматизації та інтелектуальні функції, такі як автоматичне доповнення коду, рефакторинг та підказки на основі контексту, що значно полегшує та прискорює процес розробки. Крім того, IntelliJ IDEA має широку підтримку інших інструментів та фреймворків, що робить його ідеальним вибором для розробки веб-додатків.

Мова програмування Java обрана для написання бекенду веб-додатку з ряду вагомих причин. Перш за все, Java відома своєю високою портативністю, що означає, що програми, написані на Java, можуть запускатися на будь-якій платформі, на якій є встановлена відповідна віртуальна Java-машина [29]. Це робить її ідеальним вибором для веб-додатків, оскільки вони можуть бути розгорнуті на різних серверах без необхідності великих модифікацій. Крім того,

Java має широкий екосистему та потужні бібліотеки, що дозволяють розробникам швидко та ефективно реалізовувати різноманітні функції, не створюючи їх з нуля. Це включає в себе бібліотеки для роботи з базами даних, веб-фреймворки для створення API та управління HTTP-запитами, інструменти для роботи з асинхронними завданнями та багато іншого.

Доцільність використання Java для написання бекенду веб-додатку полягає в її здатності забезпечити стабільну та ефективну роботу системи на великому обсязі даних та великій кількості запитів. Вона дозволяє розробникам швидко створювати масштабовані та надійні додатки, які можуть ефективно взаємодіяти з фронтендом, базою даних та іншими складовими системи. Популярність мови програмування для розробки бекенду веб-додатків визначається й іншими важливими перевагами, серед яких відсутність необхідності в ручному керуванні пам'яттю завдяки механізму автоматичного управління пам’яттю за допомогою garbage collector. Цей механізм звільняє розробників від необхідності в ручного виділення та звільнення пам'яті, що дозволяє уникнути багатьох типових помилок, пов'язаних з управлінням пам'яттю, таких як витоки пам'яті або помилки пов'язані з неправильним використанням пам'яті. Garbage collector відслідковує об'єкти, які більше не потрібні для програми, і автоматично видаляє їх з пам'яті, відновлюючи таким чином доступну пам'ять для нових об'єктів. Цей механізм дозволяє покращити продуктивність та стабільність веб-додатків, оскільки зменшує ймовірність виникнення помилок через недостатність пам'яті або невірне її використання.

Для створення алгоритму зчитування інформації з апаратного USB-токена використано середовище розробки Visual Studio та мова програмування C++ [30]. Це середовище розробки відоме своєю потужністю та зручністю для розробки програмного забезпечення на C++. Воно надає широкі можливості для роботи з проектами різного розміру та складності, включаючи інструменти для налагодження, аналізу коду та автоматизації процесу розробки. Однією з переваг Visual Studio є вбудована підтримка для мови C++, включаючи розширену підтримку стандартів та інтегровані інструменти для роботи з проектами. C++ відома своєю високою продуктивністю та ефективністю в роботі з ресурсами

комп'ютера. Це дозволяє написати ефективний та швидкий алгоритм зчитування, що особливо важливо при роботі зі збільшеними об'ємами даних. Крім того, дана мова має низькорівневий характер, що дозволяє працювати з пам'яттю та пристроями безпосередньо, що є важливим для роботи з флеш-носіями. Це дає більший контроль над ресурсами комп'ютера та дозволяє оптимізувати алгоритм під конкретні вимоги. Також C++ є мовою загального призначення, що означає, що вона має широкий спектр застосувань та підтримку на різних платформах. Це дозволяє створити алгоритм зчитування, який буде сумісним з різними операційними системами та пристроями.

Фреймворк Spring Boot обраний через його здатність значно спрощувати процес розробки веб-додатків, пропонуючи набір інструментів та бібліотек для реалізації різних аспектів програмного забезпечення [31]. Spring Boot дозволяє швидко створювати самостійно розгортальні додатки, що знижує витрати часу на конфігурацію та налаштування, забезпечуючи при цьому високу масштабованість та гнучкість. Завдяки цьому, розробники можуть зосередитися на логіці додатка, а не на налаштуваннях середовища.

Spring Boot складається з кількох ключових частин:

- Spring Core Container: це основа Spring Boot, яка надає реалізацію Inversion of Control (IoC) та Dependency Injection (DI) [32]. Вона включає контейнер IoC, який керує об'єктами додатку та їх взаємозв'язками, а також забезпечує їх автоматичне впровадження.
- Spring Web: ця частина надає засоби для створення веб-додатків, такі як веб-контролери, обробники запитів, підтримка шаблонів (наприклад, Thymeleaf або FreeMarker), а також можливості роботи з RESTful сервісами [33].
- Spring Data Access / Spring Data JPA: дані модулі надають інструменти для взаємодії з базами даних через JPA або інші методи доступу до даних, такі як JDBC або NoSQL [34].
- Spring Boot Starter: це набори залежностей, які автоматично налаштовуються для різних типів додатків (наприклад, веб-додатків,

додатків для зберігання даних тощо). Вони дозволяють швидко створювати додатки, використовуючи популярні стеки технологій [35].

Spring Boot ґрунтується на принципах Inversion of Control (IoC) та Dependency Injection (DI). IoC визначає, що контейнер керує життєвим циклом об'єктів та виконує їх зв'язування, тоді як DI дозволяє встановлювати залежності в об'єкти без прямого їх створення. Завдяки цим принципам розробники можуть зосередитися на реалізації бізнес-логіки, не дбаючи про створення та керування об'єктами та їх залежностями, що сприяє полегшенню розробки та підтримки додатків.

Використання Vaadin framework для написання фронтенд частини додатка має кілька значних переваг. По-перше, Vaadin надає можливість розробки веб-додатків на Java без потреби в ручному написанні HTML, CSS або JavaScript коду. Це дозволяє розробникам зосередитися на бізнес-логіці додатка, не витрачаючи час на навчання та використання різноманітних веб-технологій. Крім того, Vaadin надає широкий набір вбудованих компонентів і контейнерів, які дозволяють швидко побудувати потрібний інтерфейс користувача без необхідності вручного розташування елементів на сторінці. Це значно спрощує та прискорює процес розробки і підтримки веб-інтерфейсу. Vaadin має вбудовану підтримку асинхронного взаємодії з сервером за допомогою технології WebSocket, що дозволяє реалізувати динамічні та інтерактивні функції без перезавантаження сторінки. Це дозволяє створювати більш зручні та ефективні веб-додатки, які відповідають сучасним вимогам користувачів.

При розробці даного програмного засобу було використано інструмент контейнеризації Docker [36]. Він надає ізольоване середовище для розгортання та виконання додатків, що робить його ідеальним інструментом для розробки та тестування програмного забезпечення на різних конфігураціях без перешкоджання основній системі. Крім того, Docker дозволяє створювати зручні та переносні середовища розробки, що спрощує роботу розробників та забезпечує єдине середовище для всього колективу.

Даний засіб розробки дозволяє створювати контейнери для баз даних, що містять всі необхідні залежності та налаштування. Це робить розгортання бази даних простим та консистентним у різних середовищах, включаючи розробку,

тестування та продуктивне середовище. Крім того, використання докер-контейнерів для баз даних дозволяє легко масштабувати та управляти ними, забезпечуючи більшу гнучкість та ефективність управління даними.

3.2 Розробка програмного засобу

Успішна розробка програмного засобу вимагає не лише ґрунтовного планування та проектування, але й ефективної програмної реалізації. В даному підрозділі буде детально описано процес створення програмного, а саме буде продемонстровано та описано практичну реалізацію усіх основних алгоритмів.

Перш за все необхідно виконати початкову конфігурацію проекту та визначити основні залежності що мають бути включені до нього. Для цього було використано спеціальну веб-сторінку, створену розробниками Spring фреймворку, яка дозволяє виконати початкову конфігурацію та завантаження проекту (рис. 3.1).

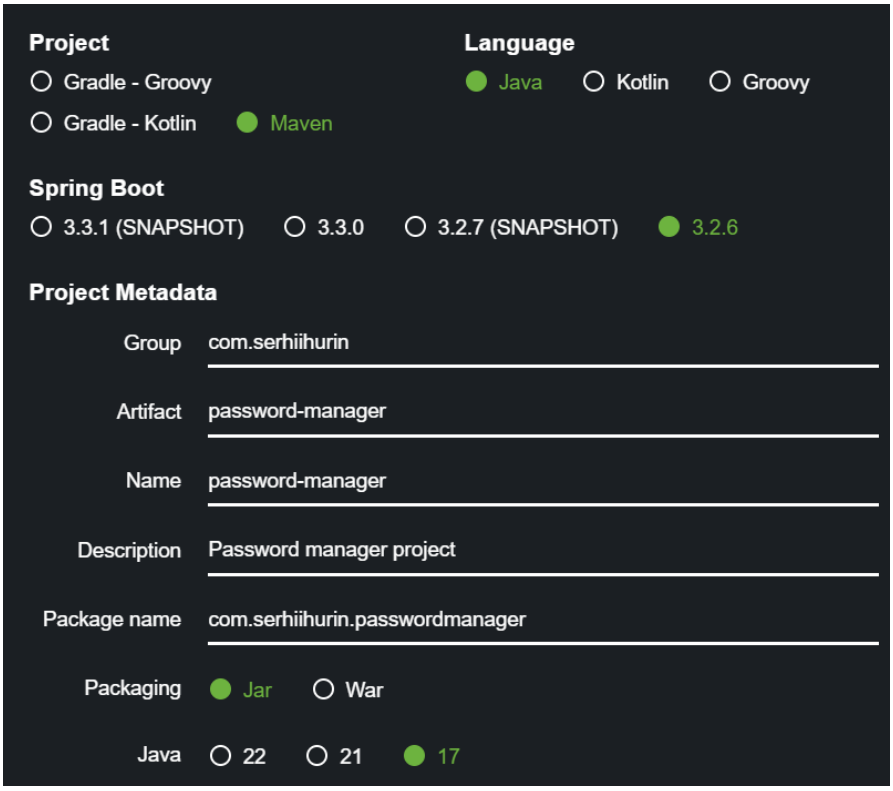


Рисунок 3.1 – Вигляд вікна конфігурації проекту

Було обрано мову програмуваня Java версії 17, SpringBoot фреймворк версії 3.2.6, а також встанолвено залежності Spring Web, Spring Security, Spring Data JPA, Vaadin, MySQL Driver для підключення до бази даних

Оскільки як засіб збирання проекту було обрано Maven [37], усі необхідні для розробки залежності та плагіни знаходяться у файлі pom.xml (рис. 3.2):

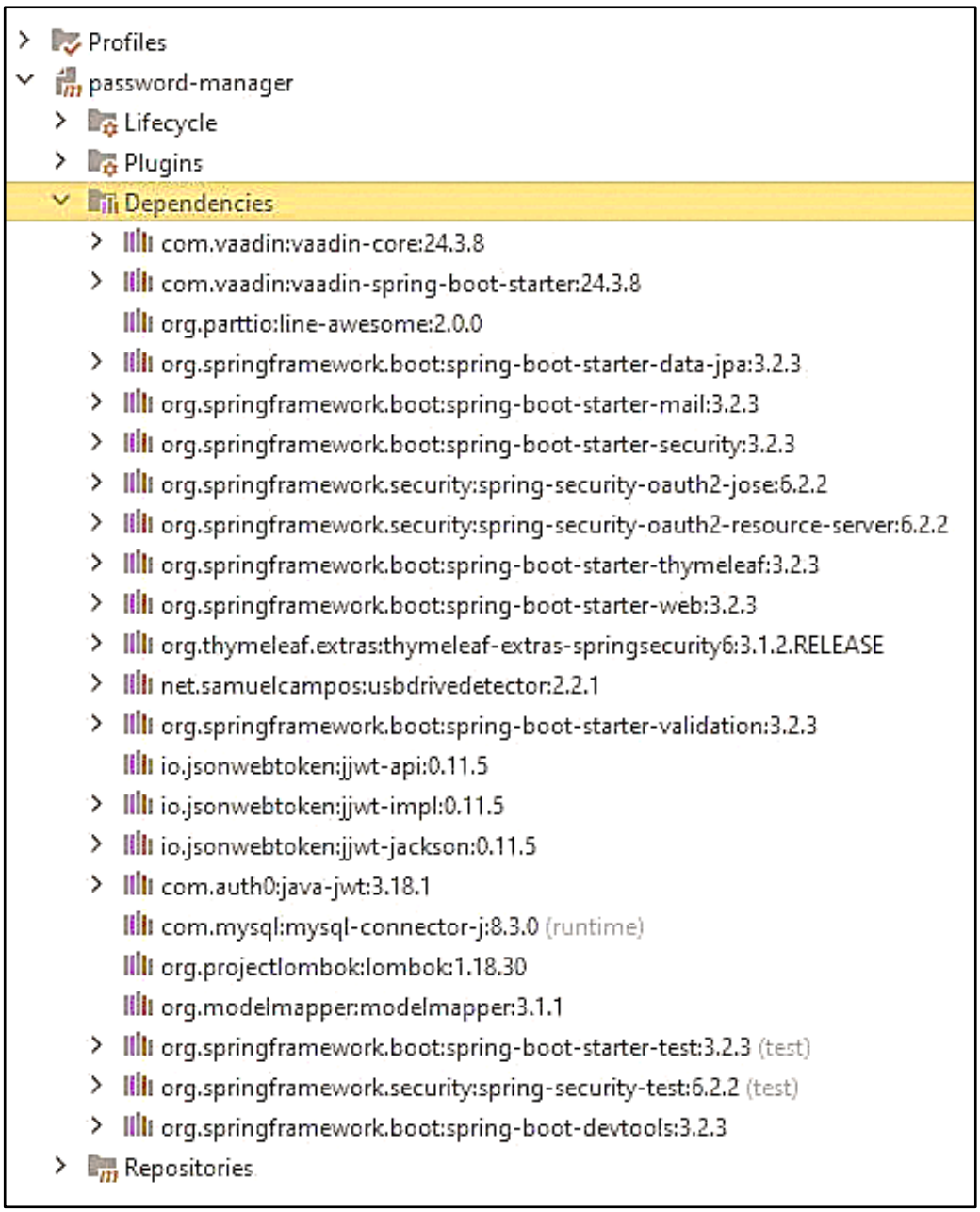


Рисунок 3.2 – Вигляд структури підключених до проекту залежностей

На рисунку відображено ключові бібліотеки, що надають увесь необхідний для розробки функціонал.

Після створення проекту необхідно налаштувати підключення до бази даних, яка буде знаходитись в Docker-контейнері. Налаштування контейнера знаходяться у файлі docker-compose.yaml. В даному файлі вказується кількість контейнерів що має бути запущено, служби які мають бути встановлені в контейнери, в даному

випадку це MySQL база даних, прив’язка порту хоста до внутрішнього порту контейнера, назва бази даних яку потрібно створити, вказання облікових даних для доступу до бази даних, таких як логін та пароль.

Налаштування підключення до бази даних, а саме посилання, за яким необхідно здійснювати підключення, дані для автентифікації в базі даних, а також налаштування порту сервера та глобальні змінні, що використовуються в програмі знаходяться у файлі application.yaml:

```
…
spring:
  application:
    name: password-manager
  datasource:
    url:
jdbc:mysql://localhost:3307/password_manager?serverTimezone=UTC
    username: ********
    password: ********
spring.jpa:
  hibernate.ddl-auto: update
ssl:
    key-store: classpath:keystore.p12
…
```

В прикладі коду варто звернути увагу на останній рядок, що вказує на файл з назвою `keystore.p12`. Для забезпечення надійної передачі даних під час комунікації через мережу було самостійно згенеровано SSL-сертифікат, що надає змогу використовувати протокол HTTPS [38].

Схематичне зображення взаємодії з базою даних наведено на рисунку 3.3 .

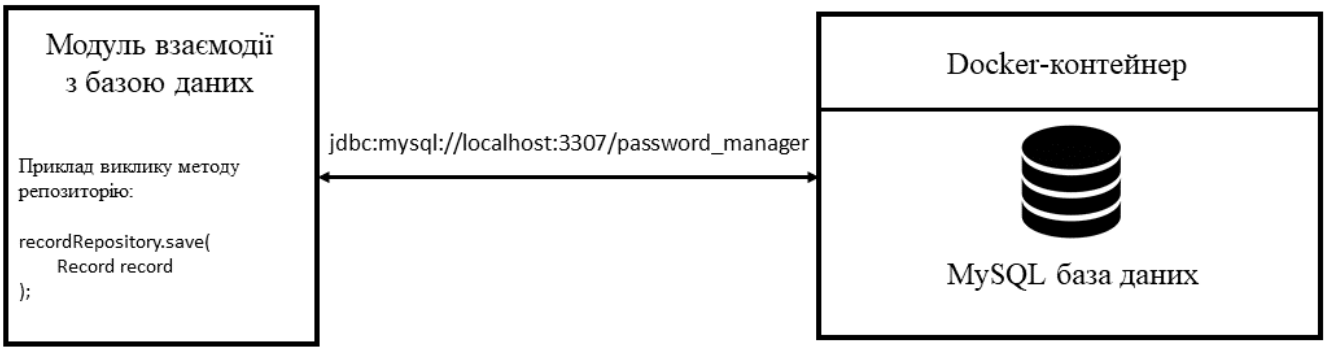


Рисунок 3.3 – Схематичне зображення взаємодії з базою даних

Вся взаємодія між додатком і базою даних, що працює в Docker-контейнері, відбувається через стандартні механізми JDBC[39] , які використовуються Spring

Data JPA. Docker-контейнер забезпечує ізольоване середовище для бази даних, а JPA репозиторії забезпечують зручний спосіб доступу до неї, абстрагуючи розробника від деталей реалізації SQL запитів.

Створення таблиць в базі даних, оновлення її структури, встановлення сесії для виконання запитів та інші операції відбуваються за допомогою фреймворку Hibernate, що є частиною модуля Spring Data JPA [34]. Для того щоб даний фреймворк створив таблиці в базі даних необхідно створити Entity-класи, які є представленням таблиць та за допомогою яких разом з DTO об’єктами відбувається взаємодія програмного засобу з базою даних [40]. Нижче наведено приклад такого класу (рис.3.4):

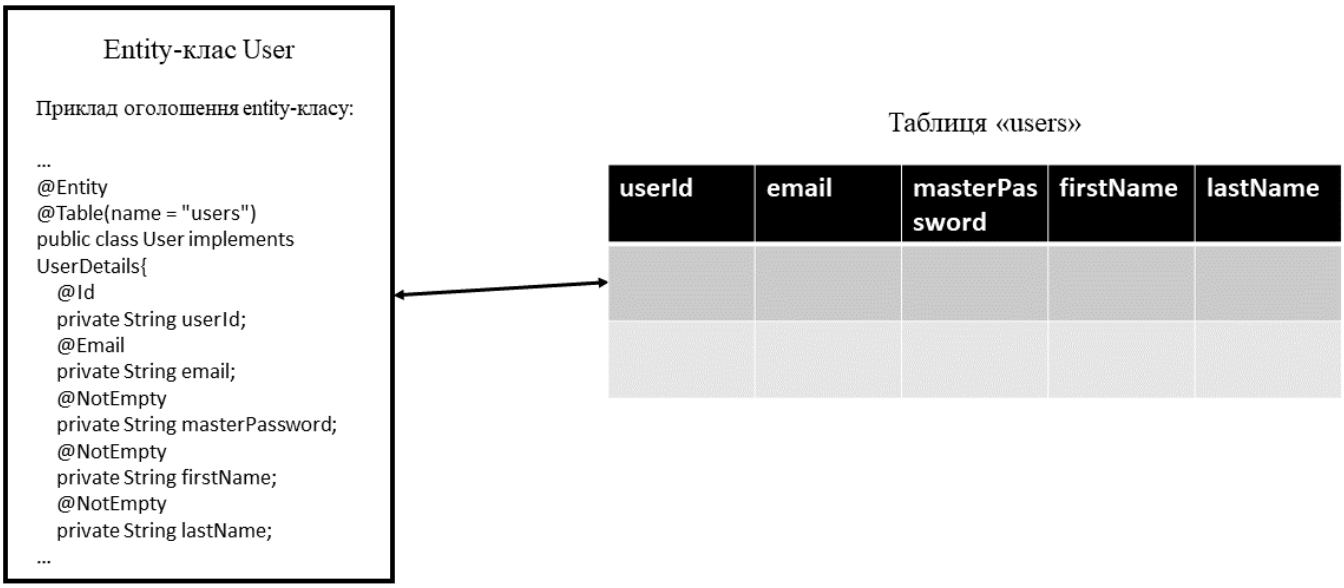


Рисунок 3.4 – Схематичне зображення оголошення Entity-класу та еквівалентної таблиці в базі даних

Анотація @Entity вказує на те, що даний клас є класом-сутністю, а @Table визначає назву таблиці що буде створено в базі даних. Аналогічим чином було створено класи для записів та груп. Схему створеної та налаштованої бази даних наведено на рисунку 3.5 .

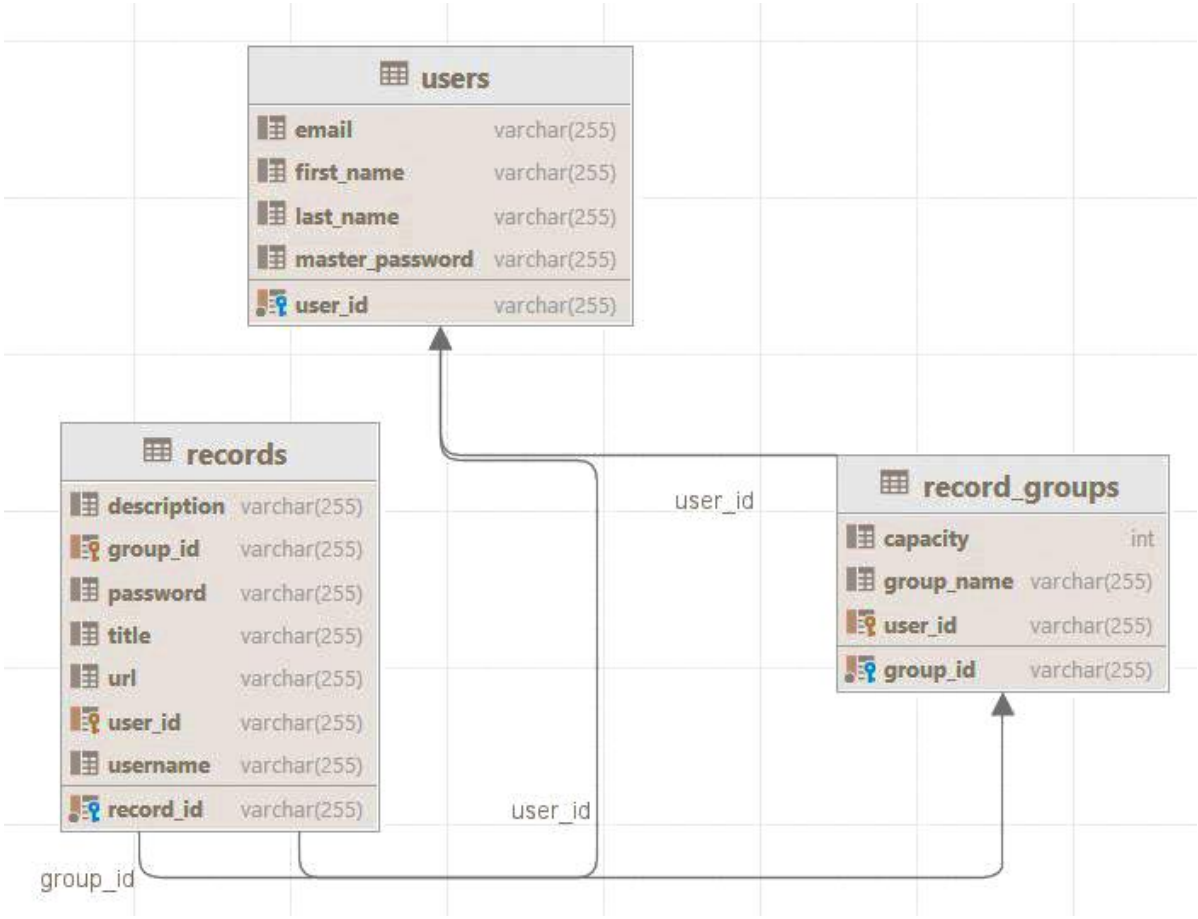


Рисунок 3.5 – Вигляд структури бази даних

Важливим етапом в розробці є налаштування безпеки веб-додатку, а саме правил створення сесії та доступу до ресурсів. Конфігурація безпеки знаходиться у класі SecurityConfig.java. У даному класі відбувається встановлення форми для автентифікації, налаштування сесії шляхом використання JWT-токену [41], додавання фільтру, що буде обробляти даний токен та перевіряти його валідність, а також налаштування тривалості сесії, що дорівнює п’ятнадцятьом хвилинам, після чого користувачеві необхідно буде повторно пройти процедуру автентифікації для подальшого доступу до функціоналу застосунку.

Після виконання усіх необхідних налаштувань, було почато розробку алгоритму зчитування інформації з апаратного токену та формування майстер-паролю. За підключення .dll бібліотеки, що містить C++ код для зчитування інформації з носія відповідає клас DllConnector.java:

Алгоритм працює циклічно, поки метод сервісу який відповідає за зчитування та передачу інформації про токен не поверне дані. Як тільки

інформацію буде отримано, алгоритм визначить літеру логічного диску який відноситься до флеш-носія та його UUID [42], а файловий сервіс, використовуючи сервіс генерації паролів створить файл в який буде поміщено згенерований ключ та збереже його на логічному диску USB-токена. Після чого, усі отримані дані, а саме назва пристрою та серійний номер, збережений у файлі ключ та UUID формують майстер-пароль від облікового запису користувача (рис. 3.6):

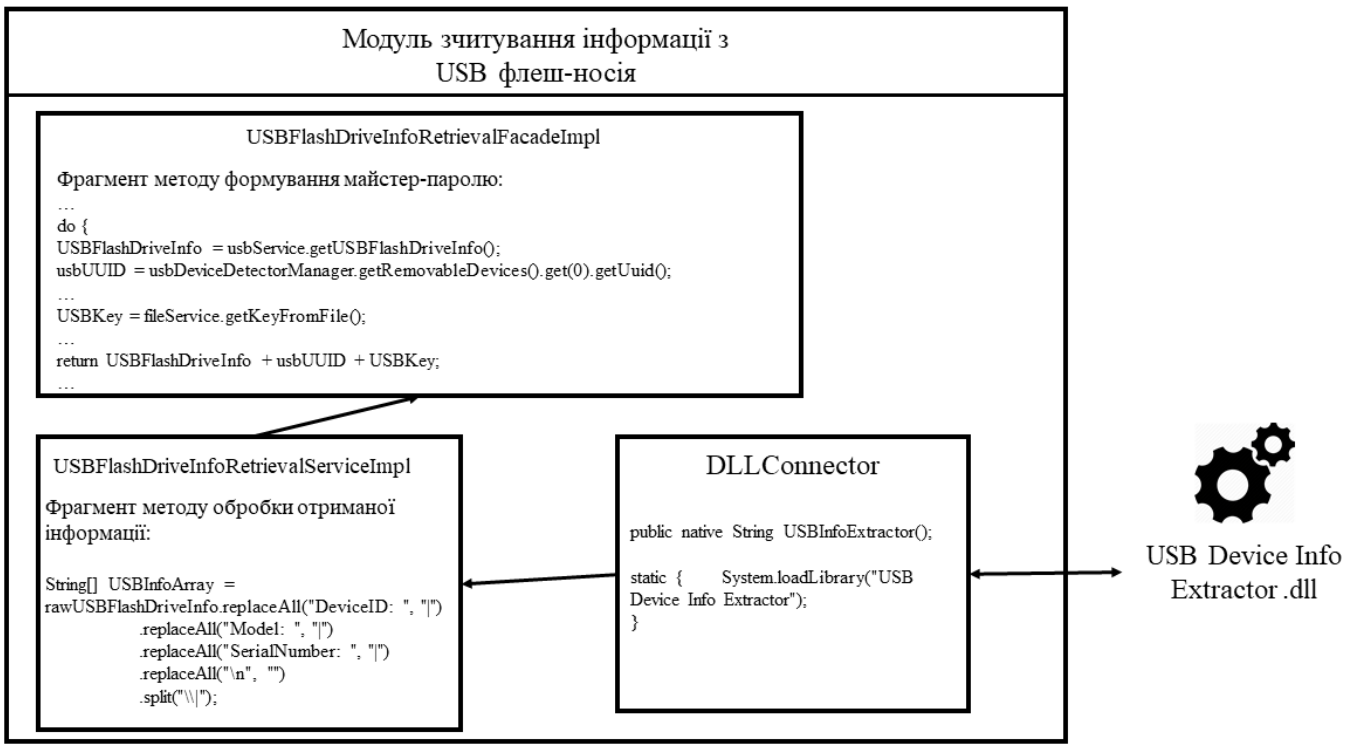


Рисунок 3.6 – Схематичне зображення реалізації схеми зчитування інформації з апаратного токену

За шифрування логінів та паролів які вводить користувач при створенні своїх записів відповідає клас EncryptionServiceImpl.java. Він використовує вбудовану бібліотеку javax.crypto яка, в даному випадку, надає реалізацію алгоритму AES.

За реалізацію алгоритму створення ключа шифрування відповідає метод `getKey(String input)` цього ж класу:

```
…
int midIndex = input.length() / 2;
String firstHalf = input.substring(0, midIndex);
String secondHalf = input.substring(midIndex);
```

…

Параметром цього методу є майстер-пароль, отриманий з бази даних. Після поділу напіл, для кожної половини окремо, викликається приватний метод `rearrangeString()`, який за допомогою звичайного циклу проводить ітерацію по усіх парних індексах символів в рядку та обмінює їх місцями з символами з непарним індексом:

```
…
for (int i = 0; i < chars.length - 1; i += 2) {
    char temp = chars[i];
    chars[i] = chars[i + 1];
    chars[i + 1] = temp;
}
…
```

Після чого, метод повертає сформований ключ, забезпечуючи коректний процес шифрування або розшифрування даних.

Генерацію надійних паролів реалізовує відповідний клас `GeneratorServiceImpl.java`. Метод, що реалізує логіку генерації приймає параметри, які передаються з інтерфейсу, а саме довжина пароля у форматі рядка та булеві змінні, які вказують на символи, що потрібно використати. Після формування алфавіту, в циклі, який проходить рівно стільки ітерацій, скільки вказано в параметрі довжини пароля, за допомогою класу SecureRandom [43] зі вказаного проміжку генерується випадковий індекс символу з алфавіту, який буде додано до пароля що формується. Клас SecureRandom у Java використовується для генерації криптографічно стійких випадкових чисел. На відміну від класу Random, який базується на псевдовипадкових числах, він забезпечує більш високий рівень випадковості та безпеки, що робить його підходящим для криптографічних застосувань. Даний клас використовує спеціалізовані алгоритми та джерела ентропії для генерації чисел, які набагато важче передбачити. Клас Random використовує лінійний конгруентний метод, який швидкий, але не забезпечує криптографічну стійкість. Тобто, при знанні початкового стану генератора можна передбачити наступні числа, що робить його непридатним для безпечного

застосування, як-от генерація ключів або паролів. SecureRandom може використовувати різні джерела випадковості, включаючи операційну систему або спеціальні апаратні модулі, для забезпечення високої ентропії. Він ініціалізується з використанням криптографічних алгоритмів, або з доступом до системних ресурсів, що збирають випадкові події, наприклад, рух миші або системні шуми.

Нижче наведено фрагмент коду з методу, що реалізує генерацію паролів:

```
…
SecureRandom random = new SecureRandom();
StringBuilder generatedPassword = new StringBuilder();
for (int i = 0; i < passwordLength; i++) {
    generatedPassword.append(
            generationCharacters.charAt(
                    random.nextInt(generationCharacters.length())
…
```

Одними з ключових компонентів розробленого програмного засобу є класи-контролери, які виконують важливі функції, такі як ініціалізація користувацького інтерфейсу та обробка запитів користувача. Ці класи об'єднують дві важливі ролі: вони створюють і налаштовують елементи інтерфейсу, а також реагують на дії користувача. Для цього використовуються механізми обробки подій (Event-driven моделі) [44]. Кожен контролер має набір прослуховувачів, які реагують на певні події, наприклад, натискання кнопок або введення тексту в поля. Коли відбувається певна подія, відповідний прослуховувач викликає методи контролера, що містять логіку обробки цієї події. Наприклад, якщо користувач натискає кнопку, прослуховувач цієї кнопки викликає метод контролера, який може виконувати певні дії, такі як обробка введених даних, виклик сервісів або оновлення інших частин інтерфейсу. Таким чином, контролери забезпечують інтерактивність додатку, дозволяючи йому відповідати на дії користувача в реальному часі. На рисунку 3.7 схематично зображено взаємодію користувача з графічним інтерфейсом за допомогою класа-контролера, який відповідає за відображення форми реєстрації:

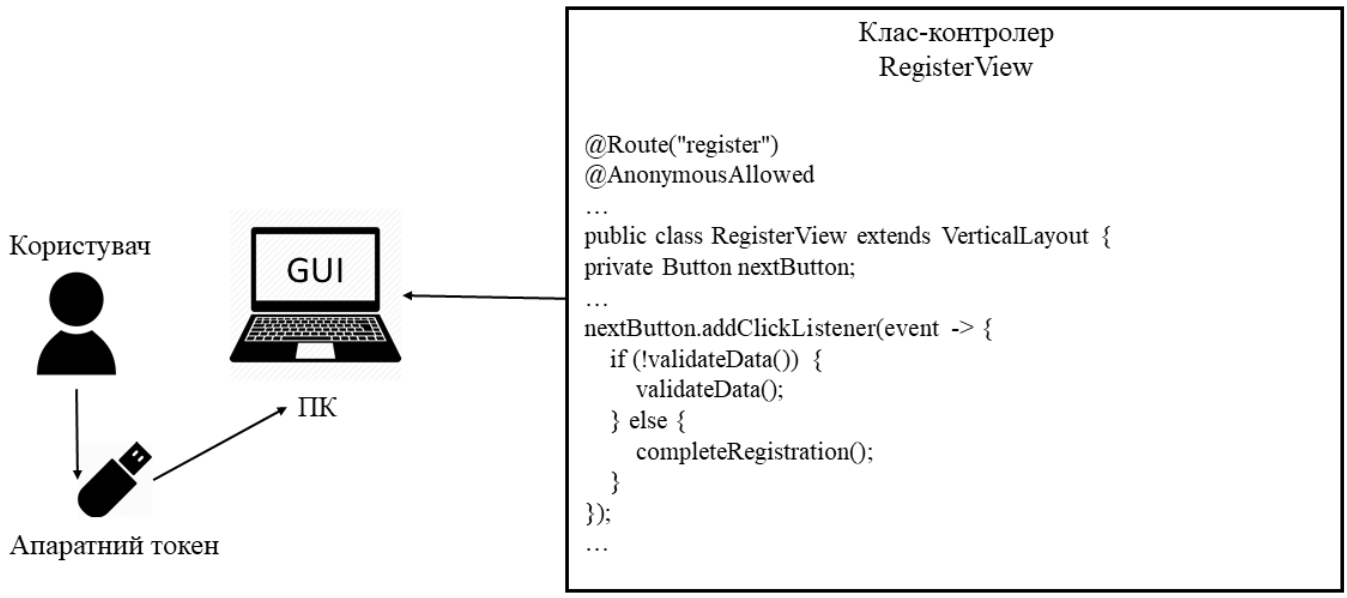


Рисунок 3.7 – Схематичне зображення реалізації класу-котролеру

На даному рисунку, анотація @Route вказує на прив’язку класу до певної URL-адреси, тобто для цього класу вона буде виглядати наступним чином:

`https://localhost:8443/register`

Де `localhost:8443` – вказує на хост та порт на якому працює сервер,

`/register` – шлях до класу, який відповідає за обробку запитів які надходять на дану URL-адресу. Надсилаючи HTTP запит за цим посиланням, буде відображено форму для реєстрації користувача.

Анотація @AnonymousAllowed вказує на те, що доступ до сторінки мають не автентифіковані, тобто анонімні користувачі.

Метод `addClickListener()` при натисканні відповідної кнопки виконує зазначену логіку, а саме відбувається валідація введених даних та виклик методу завершення реєстрації, який викликає методи сервісів, необхідних для збереження даних користувача в базі даних та подальшої автентифікації.

Таким чином, реалізувавши усі необхідні алгоритми програмного засобу та функції які він має виконувати, загальну структуру проекту що включає усі створені класи та зв’язки між ними, зображено на рисунку 3.8.

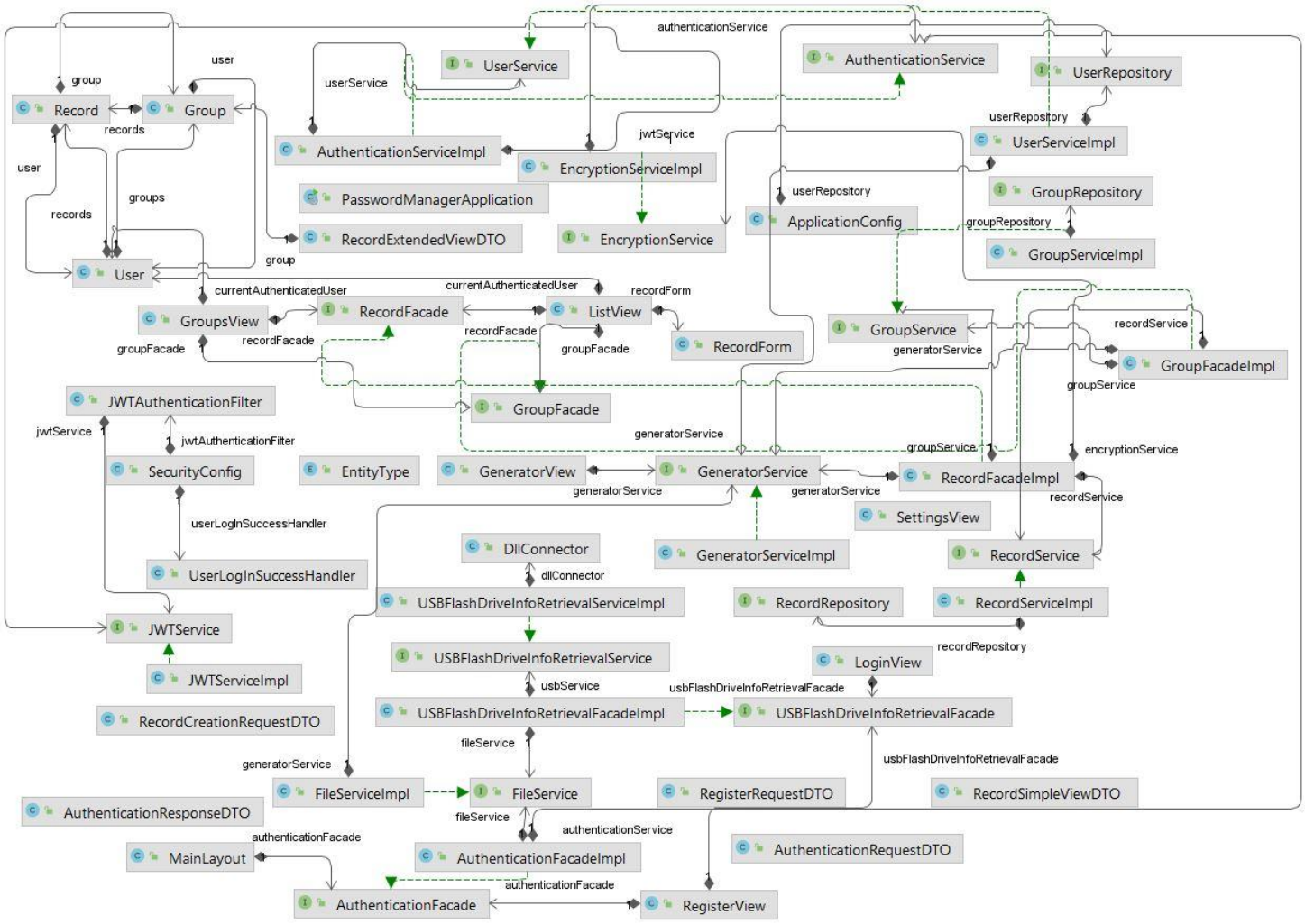


Рисунок 3.8 – Загальна структура розробленого програмного засобу та зв’язків між класами

3.3 Тестування програмного засобу

Тестування є невід'ємною частиною процесу розробки. Воно забезпечує не лише виявлення та усунення помилок, але й гарантує відповідність продукту функціональним та нефункціональним вимогам. Ефективне тестування програмного забезпечення дозволяє забезпечити високу якість продукту, підвищити його надійність, безпеку та продуктивність. В цьому розділі виконано тестування усіх основних функції засобу для керування паролями. Окрема увага приділена тестуванню безпеки, продуктивності та юзабіліті, що є критичними аспектами для програмного засобу такого типу.

Перш за все, необхідно провести тестування роботи алгоритмів реєстрації та атвентифікації, адже вони є одними з ключових елементів програмного засобу і

саме з них починається взаємодія користувача із ним. В даному тесті слід також перевірити коректність роботи алгоритму зчитування інформації з апаратного токену, валідацію даних та обробку невдалої спроби автентифікації. Перейдено за посиланням https://localhost:8443/, в результаті відкривається вкладка з формою для автентифікації (рис. 3.9).

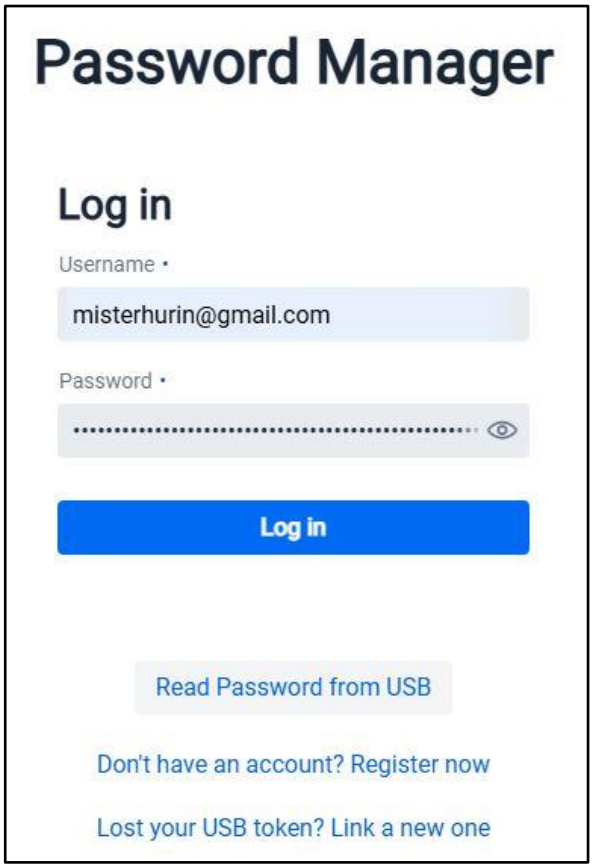


Рисунок 3.9 – Вигляд вікна автентифікації користувача

Для реєстрації користувача при першому вході до програмного засобу, необхідно натиснути на посилання під кнопкою зчитування даних з USB носія. В результаті з’являється форма для реєстрації користувача (рис. 3.10).

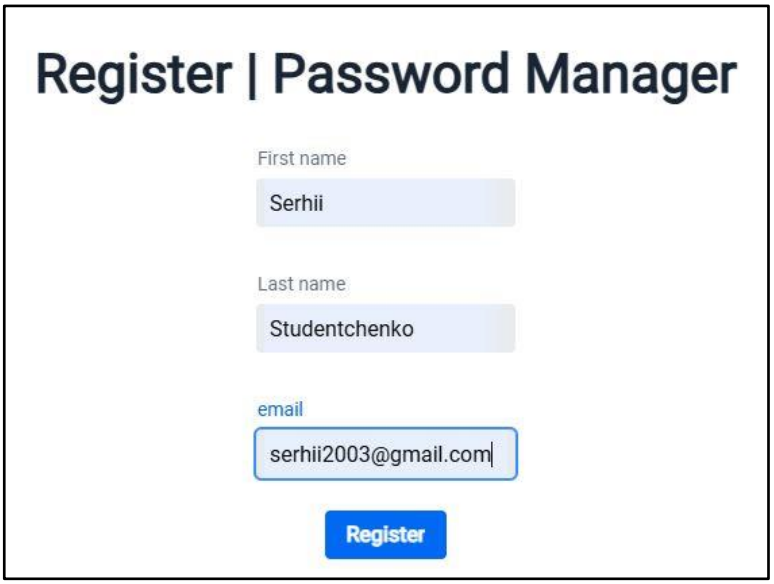


Рисунок 3.10 – Вигляд форми реєстрації користувача

Заповнено форму тестовими даними. При натисканні кнопки має відбутися зчитування інформації з токена, формування і шифрування майстер-паролю, а також створення нового запису з даними користувача в базі даних. Результати роботи алгоритму реєстрації наведено на рисунку 3.11.

```
: Got string: JetFlash Transcend 16GB USB Device 2029607934
: USBStorageDevice(rootDirectory=E:\, deviceName=ESD-USB, device=E:\, uuid=58F50949)
: Initial password JetFlash Transcend 16GB USB Device 202960793458F5094902oyx0!d2bCgouuc$*c#g-Md@J+q
: Triggered authentication method for user: serhii2003@gmail.com with password: JetFlash Transcend 1
: Authenticated user in ListView: User(userId=uid_937GermQN8qA0yIfC5nw, email=serhii2003@gmail.com,
: $2a$10$uby/RqarsyaVfDn110yYOejKAqBYKOdgqUZYVPjOi04hDsyCcXqCu
```

Рисунок 3.11 – Зображення логів, створених під час реєстрації користувача

При реєстрації зчитано назву та серійний номер носія, далі визначено літеру логічного диску, що асоційований з носієм, на який слід зберегти файл зі згенерованим за допомогою сервісу генерації ключем. Окрім того, отримано UUID носія, успішно пройдено автентифікацію та проведено шифрування сформованого майстер-паролю перед збереженням до бази даних (рис 3.12).

ORDER BY

email	first_name	last_name	master_password
serhii2003@gmail.com	Serhii	Studentchenko	$2a$10$uby/RqarsyaVfDn110yYOejKAq…

Рисунок 3.12 – Зображення стану бази даних після реєстрації користувача

Як видно з рисунку, дані користувача які було введено при реєстрації успішно збережено. При спробі залишити будь-яке поле в формі реєстрації пустим має бути відображено відповідне сповіщення у вкладці браузера. Поле з ім’ям залишено пустим та натиснуто кнопку завершення реєстрації (рис 3.13).

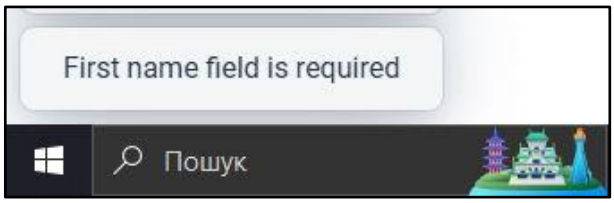


Рисунок 3.13 – Зображення спливаючого сповіщення про помилку під час валідації

Протестовано процедуру автентифікації користувача. При введенні правильного логіну та використання відповідного токену, після натискання кнопки автентифікації, отримано логи зі зчитаною інформацією про флеш-носій (рис 3.14). Як тільки завершилася робота алгоритму автентифікації, було отримано доступ до інтерфейсу менеджера паролів.

```
: Got string: EXU2064GB USB Device 3771901030876049542
: USBStorageDevice(rootDirectory=E:\, deviceName=USB-дисковод, device=E:\, uuid=16653E22)
```

Рисунок 3.14 – Зображення логів, створених під час автентифікації користувача

При спробі підключення стороннього флеш-носія, або ж введені хибного логіну, при натисканні кнопки автентифікації буде виведено спливаюче сповіщення про невідповідність даних, при цьому не відбудеться жодних змін на вкладці автентифікації та користувач не зможе отримати доступ до функціоналу програмного засобу.

Таким чином, протестовано алгоритми реєстрації, автентифікації користувачів та зчитування інформації з апаратного токена. Жодних проблем в роботі виявлено не було, усі алгоритми виконують свої функції коректно.

Наступним кроком виконано тестування алгоритмів взаємодії з базою даних та шифрування конфіденційної інформації. Користувач повинен мати змогу виконувати операції читання, запису, редагування та видалення записів. При

цьому, така інформація як логін та пароль при збереженні до бази даних має бути зашифрована, а за необхідності коректно розшифрована. Створення запису відбувається за допомогою натискання відповідної кнопки на головній сторінці програмного засобу, яка відображає форму для введення необхідних даних (рис. 3.15).

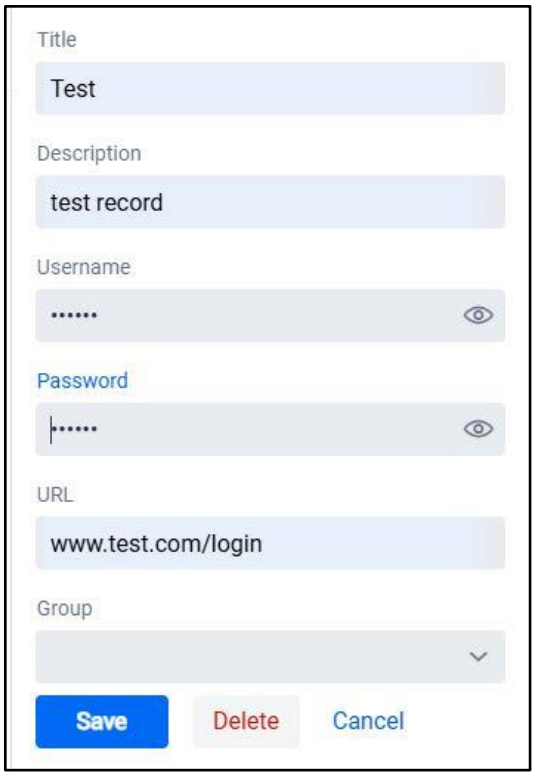


Рисунок 3.15 – Вигляд форми створення запису

Поля з логіном та паролем автоматично приховують вміст, тому слід зазначити, що при створені даного запису було використано логін «123abc» та пароль «qwerty». При натисканні кнопки «Зберегти» було збережено введену інформацію до бази даних та попередньо зашифровано логін та пароль (рис. 3.16).

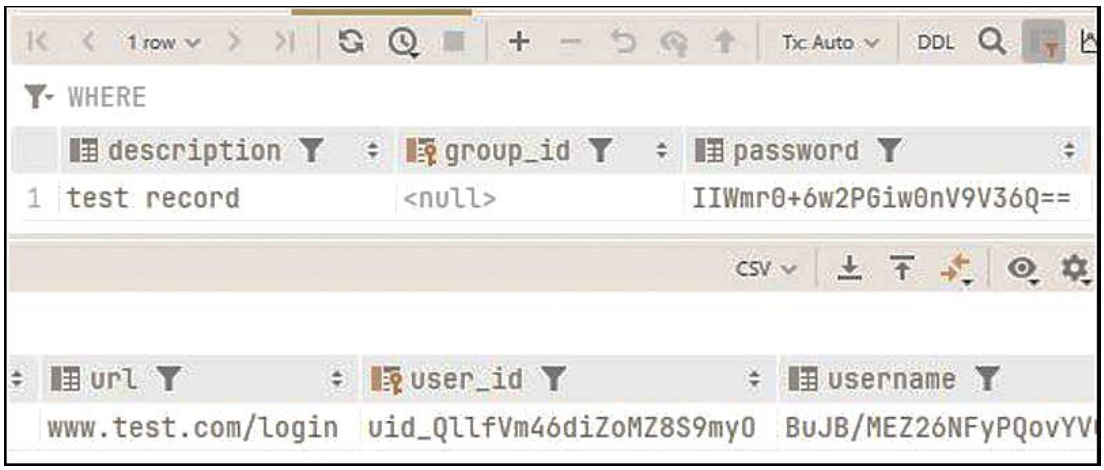


Рисунок 3.16 – Вигляд форми реєстрації користувача

Як можна побачити з рисунку, поля в стовпцях паролю та логіну збережено в зашифрованому вигляді.

Отримано інформацію з бази даних та перевірено коректність розшифрування даних (рис. 3.17).

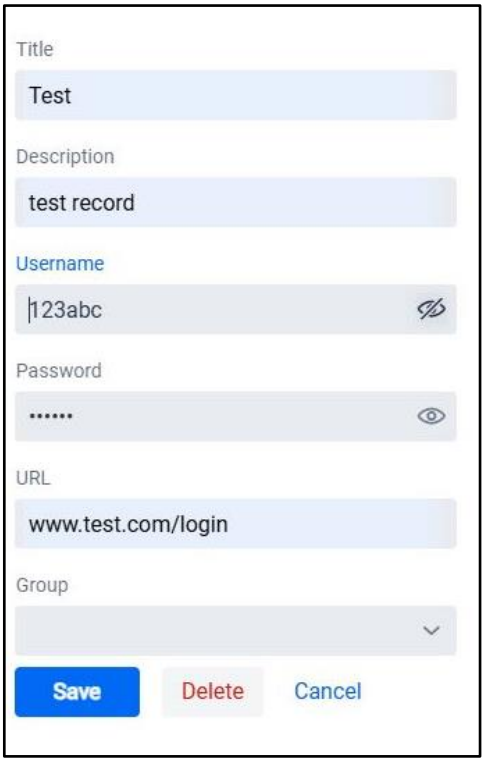


Рисунок 3.17 – Демонстрація коректного розшифрування інформації після повернення з бази даних

Логін користувача як і пароль було коректно розшифровано, що означає, що користувач зможе легко скористатися своїми обліковими даними при

автентифікації на сторонньому ресурсі, або ж відредагувати та повторно зберегти їх за потреби.

Додано декілька нових записів (рис. 3.18) та протестовано можливість їх пошуку та фільтрації за назвою. Для цього потрібно скористатися відповідним полем на головній сторінці менеджера паролів, що знаходиться поруч з кнопкою додавання нового запису.

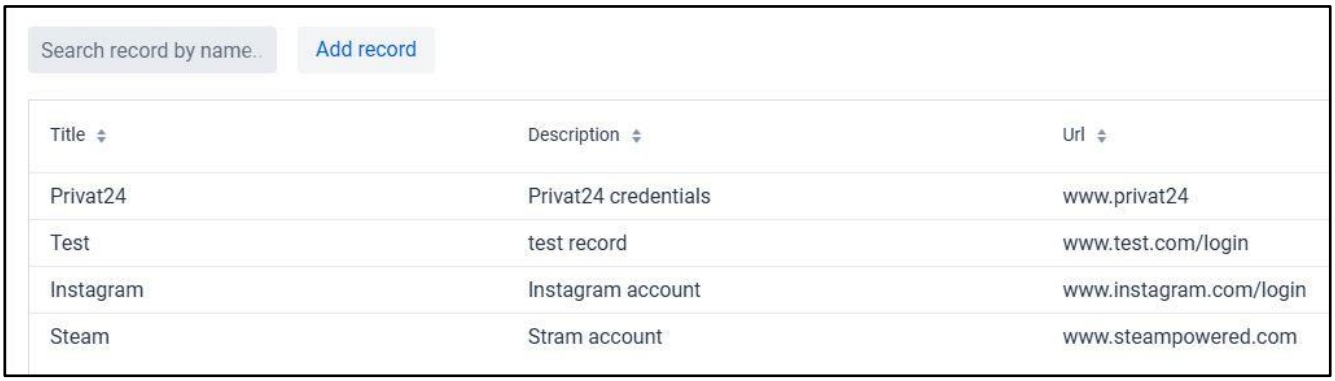


Рисунок 3.18 – Вигляд частини головної сторінки з відображенням наявних у користувача записів

Коли користувач починає вводити у поле пошуку повну або часткову назву, список записів має бути відфільтровано таким чином, щоб залишилися лише ті записи, які відповідають заданому патерну. Поле пошуку є нечутливим до регістру символів. Результат виконання пошуку наведено на рисунку (3.19).

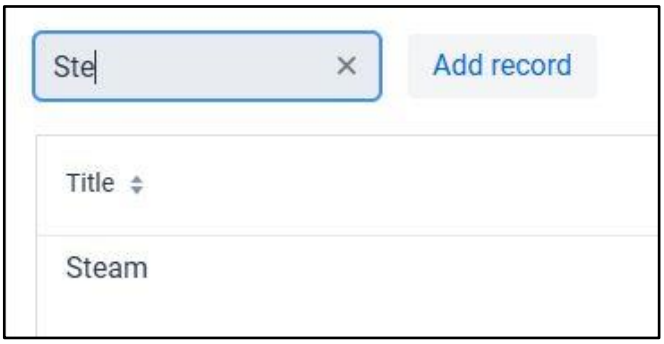


Рисунок 3.19 – Зображення відфільтрованих за назвою записів

Здійснено видалення тестового запису. В результаті при натисканні кнопки видалення інформація про запис зникла як з інтерфейсу користувача, так і з бази даних (рис. 3.20).

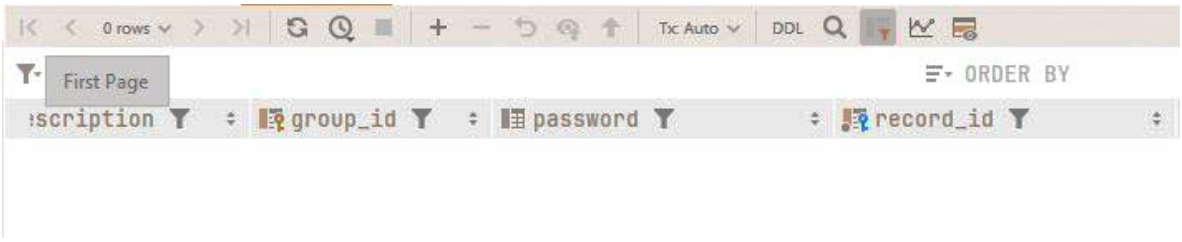


Рисунок 3.20 – Демонстрація відсутності інформації в базі даних після видалення запису

Наступним етапом тестування буде перевірка коректності роботи алгоритму генерації надійних паролів. Згенерований пароль повинен відповідати вказаним за допомогою інтерфейсу параметрам. Також буде виконано перевірку на надійність за допомогою спеціалізованого веб-сервісу.

Кожен раз при переході на сторінку генератора паролів має генеруватися новий пароль за вказаними за замовчуванням параметрами, а саме наявність літер верхнього та нижнього реєстру, цифр та спецсимволів, а також бути довжиною в п’ятнадцять символів (рис. 3.21).

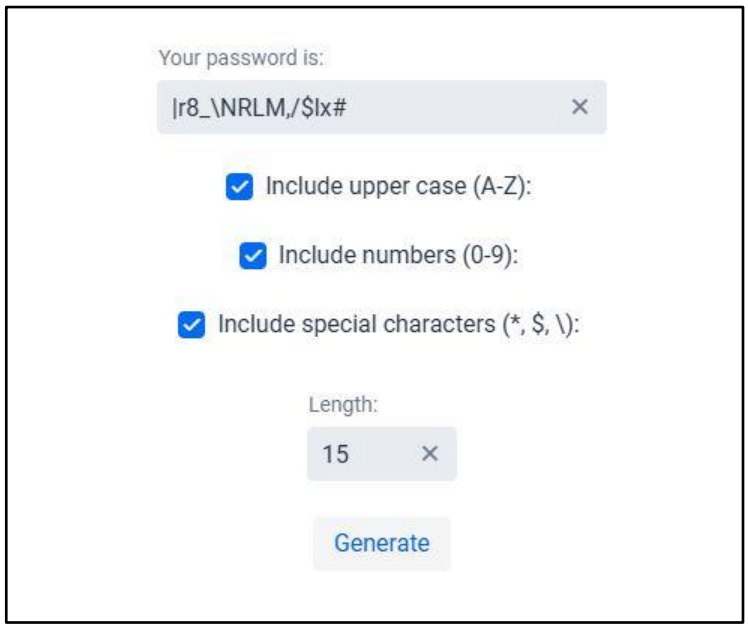


Рисунок 3.21 – Вигляд вікна генератора паролів зі встановленими за замовчуванням параметрами

Перевірено щойно згенерований за допомогою алгоритму пароль за допомогою сервісу security.org [45]. Результат перевірки наведено на рисунку (3.22).

Рисунок 3.22 – Зображення вікна сервісу для перевірки надійності згенерованого пароля

Як видно на рисунку, для взлому паролю, згенерованого за встановленими за замовчуванням значеннями, комп'ютеру використовуючи метод грубої сили знадобиться чотириста мільярдів років, що безсумнівно є гарним результатом.

Перевірено можливість, в разі виникнення такої необхідності, згенерувати пароль не використовуючи спецсимволи. Результат виконання тесту зображено на рисунку 3.23.

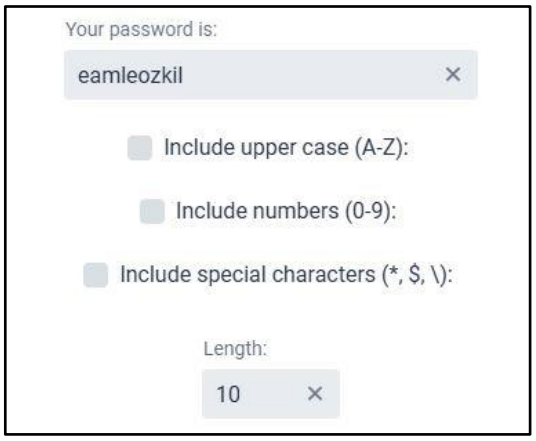


Рисунок 3.23 – Вигляд вікна генератора паролів зі встановленими користувачем параметрами

В даному випадку було вказано не використовувати жодних додаткових символьних алфавітів, тому було згенеровано пароль вказаної довжини використовуючи латинські літери нижнього регістру, що є алфавітом за

замовчуванням. Отже, даний алгоритм працює коректно, генерація паролів відбувається належним чином, проблем при роботі не виявлено.

Протестовано роботу алгоритму прив’язки до нового апаратного токена у разі втрати доступу до старого. Для цього створено нового тестового користувача та один тестовий запис. Початкові дані запису наведено на рисунку 3.24 .

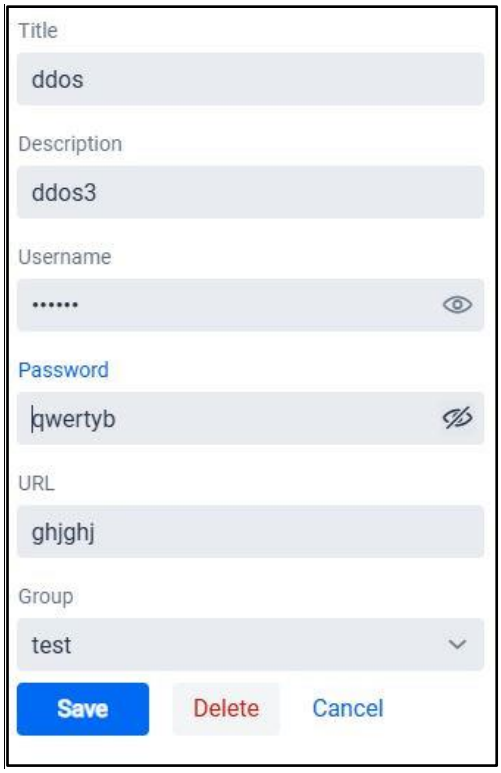


Рисунок 3.24 – Зображення початкових даних тестового запису перед виконанням тесту на прив’язку до нового токена

Логін та пароль збережено у базі даних в зашифрованому вигляді (рис. 3.25).

description	group_id	password
ddos3	gid_PX47sJNTTfNj41LPvITq	WPqlQ3J+gK8jR+z9pRZ+fQ==

Рисунок 3.25 – Відображення стану запису в базі даних перед виконанням тесту на прив’язку до нового токена

Відкрито вкладку налаштувань та обрано пункт «Змінити токен». В результаті відкрито меню, в якому необхідно вказати електронну адресу користувача яка є логіном від акаунта в менеджері паролів, ввести отриманий код верифікації та підключити новий USB-носій до якого буде виконано прив’язку (рис.3.26).

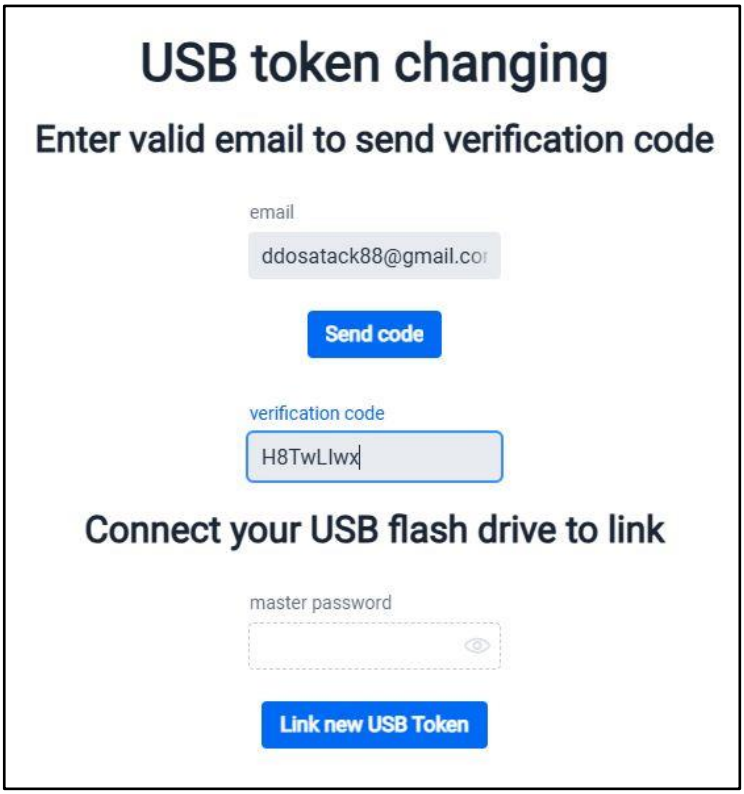


Рисунок 3.26 – Відображення меню прив’язки до нового апаратного токена

Перевірено обробку введення не відповідної електронної пошти, тобто такої, яка не прив’язана до облікового запису. Внаслідок чого отримано сповіщення про помилку (рис. 3.27).

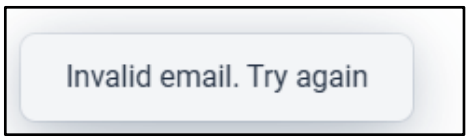


Рисунок 3.27 – Зображення сповіщення про невідповідну електронну пошту

Після натискання кнопки відправки коду верифікації на коректну електронну пошту, отримано лист з кодом та інструкціями (рис. 3.28).

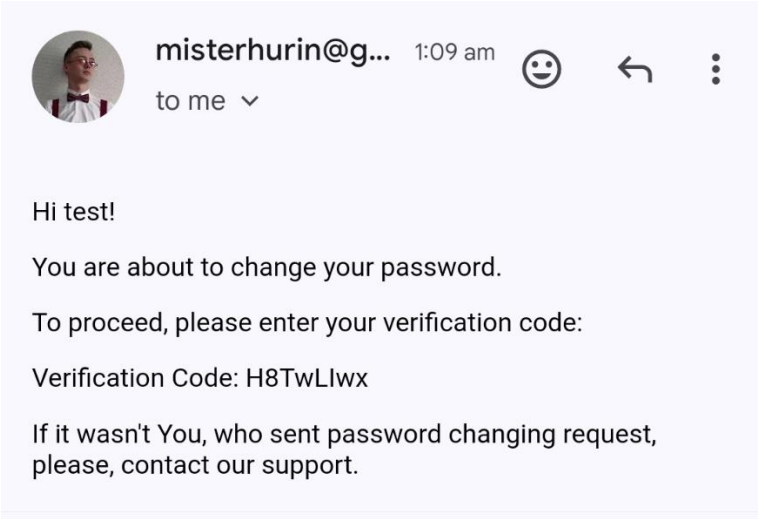


Рисунок 3.28 – Відображення вмісту листа з кодом верифікації відправленого на електронну пошту

При введенні невірного коду виведено сповіщення про помилку та здійснено переадресацію на головну сторінку. У випадку введення коректного коду верифікації, при підключенні USB-носія виведено сповіщення про успішну заміну майстер-пароля та виконано переадресацію на головну сторінку (рис. 3.29).

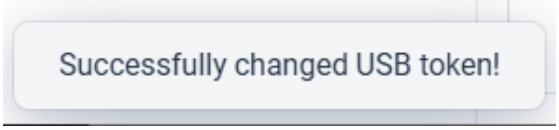


Рисунок 3.29 – Зображення сповіщення про успішну заміну токена

При цьому розшифровано та повторно зашифровано усі логіни та паролі тестового користувача (рис. 3.30).

description	group_id	password
ddos3	gid_PX47sJNTTfNj41LPvITq	rJbfGHpqxM0u7O3ej7EN9g==

Рисунок 3.30 – Відображення стану запису в базі даних після виконання тесту на прив’язку до нового токена

Як видно на рисунку, зашифроване значення пароля тестового запису змінилося в порівнянні з тим, що було до проведення тестування роботи алгоритму, адже при повторному шифруванні використано новий майстер-пароль, при цьому

дані запису отримано в коректному форматі, вихідне значення розшифрованого логіна та пароля не змінилося, отже дотримано цілісності даних.

3.4 Висновки до третього розділу

Таким чином, в даному розділі було виконано роботу, яка охоплює всі етапи створення ефективного та безпечного менеджера паролів на основі апаратного токену. В першу чергу було проведено обґрунтування вибору інструментальних засобів розробки, детально описано програмну реалізацію, що включала конфігурацію проекту, реалізацію основних алгоритмів та функцій. Показано загальну структуру додатку, яка включає усі розроблені класи та відображає їхню взаємодію. Конфігурація проекту включала налаштування середовища розробки, залежностей та інтеграцію з Docker для контейнеризації додатку. Реалізація основних алгоритмів та функцій була спрямована на забезпечення надійної автентифікації користувачів та зберігання паролів з використанням апаратного токену. Проведено тестування розробленого програмного засобу, яке охоплювало як функціональні аспекти, так і безпекові, що дозволило виявити та усунути можливі недоліки.

ВИСНОВКИ

У бакалаврській дипломній роботі було виконано завдання, що забезпечили створення ефективного та безпечного засобу керування паролями на основі апаратного токену. Проведено аналіз існуючих засобів керування паролями, їх переваг та недоліків, що дозволило виявити ключові вимоги та напрямки для вдосконалення. Досліджено сучасні технології та стандарти апаратних токенів, зокрема Google Titan, OnlyKey та YubiKey, що дало змогу обрати найефективніші рішення для реалізації високого рівня безпеки.

Розроблено архітектуру програмного засобу та реалізовано механізми реєстрації та автентифікації за допомогою апаратного токену, що суттєво підвищує захищеність системи. Забезпечено шифрування даних для збереження їхньої конфіденційності та захисту від несанкціонованого доступу, що є критично важливим для зберігання та управління паролями.

Проектування та розробка програмного засобу включали створення його структури та алгоритмів, що дозволили інтегрувати апаратні токени для автентифікації користувачів. Найголовнішим досягненням роботи стало покращення методів реєстрації та автентифікації шляхом прив'язки до даних USB-флеш носія, що спрощує користувацький досвід, не погіршуючи, а навіть покращуючи рівень безпеки.

Проведене тестування програмного засобу підтвердило його надійність і стабільність, а також ефективність у виконанні своїх функцій. Результати тестування демонструють відповідність програмного засобу визначеним вимогам та стандартам, забезпечуючи високий рівень безпеки та зручності для користувачів.

Таким чином, виконана робота не лише підтвердила доцільність використання апаратних токенів у менеджерах паролів, але й продемонструвала, що такі рішення можуть значно підвищити безпеку та зручність для користувачів. Розроблений програмний засіб має потенціал для подальшого вдосконалення та широкого впровадження у практику, забезпечуючи надійне та безпечне керування паролями.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is a Password? URL: https://www.beyondtrust.com /resources/glossary/password (date of access: 23.05.2024).

2. What is Biometric Authentication and How Does It Work? URL: https://www.logintc.com/types-of-authentication/biometric-authentication/#:~:text=Biometric%20authentication%20refers%20to%20a,,%20retinas,%20and%20facial%20features (date of access: 07.05.2024).

3. Creating Strong Passwords: Importance and Best Practices. URL: https://www.eccu.edu/blog/technology/the-importance-of-strong-secure-passwords/ (date of access: 07.05.2024).

4. Baker K. What is Mobile Malware? Types & Prevention Tips - CrowdStrike. URL: https://www.crowdstrike.com/cybersecurity-101/malware/mobile-malware/ (date of access: 08.05.2024).

5. Kerner S. M. What is a password manager? URL: https://www.techtarget.com/searchsecurity/definition/password-manager (date of access: 10.05.2024).

6. Glamoslija K. Bitwarden review 2024 – open-source, but is it good? URL: https://www.safetydetectives.com/best-password-managers/bitwarden/ (date of access: 10.05.2024).

7. Jennings M. LastPass review: pros & cons, features, ratings, pricing and more. URL: https://www.techradar.com/reviews/lastpass (date of access: 10.05.2024).

8. Long E. 1Password password manager review. URL: https://www.tomsguide.com/reviews/1password (date of access: 10.05.2024).

9. Kallstrom G. KeePass Review 2024: Expert Rated 3.6/5 Password Manager. URL: https://www.passwordmanager.com/keepass-review/ (date of access: 10.05.2024).

10. LastPass Hacked - Identified Early & Resolved - The LastPass Blog. URL: https://blog.lastpass.com/posts/2015/06/lastpass-security-notice (date of access: 11.05.2024).

11. Trevino A., Cutler A., Guccione D. What is a Hardware Security Key and How Does It Work? Keeper Security Blog - Cybersecurity News & Product Updates. URL: https://www.keepersecurity .com/blog/2023/05/09/what-is-a-hardware-security-key-and-how-does-it-work/ (date of access: 16.05.2024).

12. Universal 2nd Factor (U2F) Overview. URL: https://fidoalliance.org/specs/u2f-specs-master/fido-u2f-overview.html (date of access: 18.05.2024).

13. FIDO2 Credentials. Yubico Product Documentation – Yubico Product Documentation documentation. URL: https://docs.yubico.com/yesdk/users-manual/application-fido2/fido2-credentials.html (date of access: 19.05.2024).

14. What is Token Authentication and How Does It Work? URL: https://www.logintc.com/types-of-authentication/token-authentication/#:~:text=One-time%20password%20(OTP),or%20receive%20one-time%20passwords (date of access: 18.05.2024).

15. Personal Identity Verification (PIV) of Federal Employees and Contractors. Gaithersburg, MD : National Institute of Standards and Technology, 2022. DOI: https://doi.org/10.6028/nist.fips.201-3 (date of access: 18.05.2024).

16. Rad R. Power BI REST API. Pro Power BI Architecture. Berkeley, CA, 2023. P. 485–493. DOI: https://doi.org/10.1007/978-1-4842-9538-0_26 (date of access: 20.05.2024).

17. Deinum M. Spring Security. Spring Boot 3 Recipes. Berkeley, CA, 2024. P. 235–277. DOI: https://doi.org/10.1007/979-8-8688-0113-6_5 (date of access: 20.05.2024).

18. Гурін С. Каплун В. Свідоцтво про реєстрацію авторського права на твір №118433. Комп`ютерна програма «ADS_Protect». Дата реєстрації 25 квітня 2023 р.

19. Гурін С. Захист програмного коду з використанням альтернативних потоків NTFS [Електронний ресурс] / С. Гурін, В. А. Каплун // Матеріали LII Науково-технічної конференції підрозділів ВНТУ, Вінниця, 21-23 червня 2023 р. –

Електрон. текст. дані. – 2023. – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2023/paper/view/17612.

20. Spring with JPA / P. Späth et al. Pro Spring 6 with Kotlin. Berkeley, CA, 2023. P. 399–435. DOI: https://doi.org/10.1007/978-1-4842-9557-1_8 (date of access: 24.05.2024).

21. Фасад. Refactoring and Design Patterns. URL: https://refactoring.guru/uk/design-patterns/facade (дата звернення: 29.05.2024).

22. Surapunt T., Doungsa-Ard C. The hibernate framework query mechanisms comparison. Lecture notes on software engineering. 2014. Vol. 2, no. 3. P. 268–274. DOI: https://doi.org/10.7763/lnse.2014.v2.135 (date of access: 01.06.2024).

23. Aibin M. The DTO pattern (data transfer object). URL: https://www.baeldung.com/java-dto-pattern (date of access: 05.06.2024).

24. ModelMapper - getting started. URL: https://modelmapper.org/getting-started/#setting-up (date of access: 05.06.2024).

25. BCryptPasswordEncoder (spring-security-docs 6.3.0 API). URL: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoder.html (date of access: 05.06.2024).

26. Yang H. Blowfish Cipher Tutorials - Herong's Tutorial Examples. Independently Published, 2018.

27. Bulus A., Bulus E. Cipher with AES. 2018 3rd International Conference on Computer Science and Engineering (UBMK), Sarajevo, 20–23 September 2018. 2018. DOI: https://doi.org/10.1109/ubmk.2018.8566543 (date of access: 06.06.2024).

28. IntelliJ IDEA – the Leading Java and Kotlin IDE. URL: https://www.jetbrains.com/idea/ (date of access: 06.06.2024).

29. Java virtual machine. Computers & Mathematics with Applications. 1997. Vol. 34, no. 10. P. 135. DOI: https://doi.org/10.1016/s0898-1221(97)90189-9 (date of access: 06.06.2024).

30. Academy C. P. C++: The Ultimate Crash Course to Learn C Plus Plus with Practical Computer Coding Exercises. Independently Published, 2019.

31. Gutierrez F. Spring Boot, Simplifying Everything. Introducing Spring Framework. Berkeley, CA, 2014. P. 263–276. URL: https://doi.org/10.1007/978-1-4302-6533-7_19 (date of access: 06.06.2024).

32. Thakre R. Mastering Dependency Injection and Inversion of Control in Spring Boot Microservices. Medium. URL: https://medium.com/@renukathakre2468/mastering-dependency-injection-and-inversion-of-control-in-spring-boot-microservices-884525fe134b (date of access: 06.06.2024).

33. Ottinger J. B., Lombardi A. Spring Web. Beginning Spring 5. Berkeley, CA, 2019. P. 145–167. DOI: https://doi.org/10.1007/978-1-4842-4486-9_6 (date of access: 06.06.2024).

34. Tudose C., Odubasteanu C. Object-relational Mapping Using JPA, Hibernate and Spring Data JPA. 2021 23rd International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, 26–28 May 2021. 2021. DOI: https://doi.org/10.1109/cscs52396.2021.00076 (date of access: 06.06.2024).

35. Spring Boot - Starters - GeeksforGeeks. URL: https://www.geeksforgeeks.org/spring-boot-starters/ (date of access: 06.06.2024).

36. Susnjara S., Smalley I. What Is Docker?. URL: https://www.ibm.com/topics/docker (date of access: 07.06.2024).

37. Collins T. What is Maven in Java? (Framework and Uses). URL: https://www.browserstack.com/guide/what-is-maven-in-java (date of access: 08.06.2024).

38. What is an SSL Certificate?. URL: https://www.digicert.com/what-is-an-ssl-certificate (date of access: 08.06.2024).

39. Piwowarek G. Introduction to JDBC. URL: https://www.baeldung.com/java-jdbc (date of access: 09.06.2024).

40. Accessing Data with JPA. URL: https://spring.io/guides/gs/accessing-data-jpa (date of access: 10.06.2024).

41. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC Editor, 2015. URL: https://doi.org/10.17487/rfc7519 (date of access: 11.06.2024).

42. Java UUID. URL: https://www.javatpoint.com/java-uuid (date of access: 11.06.2024).

43. Random vs Secure Random numbers in Java. URL: https://www.geeksforgeeks.org/random-vs-secure-random-numbers-java/ (date of access: 11.06.2024).

44. Codex A. C. What is event-driven programming and how is it used in JavaScript?. URL: https://reintech.io/blog/what-is-event-driven-programming-in-javascript (date of access: 12.06.2024).

45. How Secure Is My Password? Password Strength Checker. URL: https://www.security.org/how-secure-is-my-password/ (date of access: 12.06.2024).

ДОДАТКИ

Додаток А

65

Протокол перевірки
бакалаврської дипломної роботи
на наявність текстових запозичень

Назва роботи: Засіб для керування паролями на основі апаратного токена

Автор роботи: Гурін Сергій В'ячеславович

Тип роботи: бакалаврська дипломна робота

Підрозділ кафедра захисту інформації ФІТКІ
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність – 98,5%. Схожість – 1,3%.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Валентина КАПЛУН
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Сергій ГУРІН
(підпис) (прізвище,ініціали)

Керівник роботи _____ Олеся ВОЙТОВИЧ
(підпис) (прізвище, ініціали)

3.5

Додаток Б

Код програмного засобу

dllmain.cpp (C++ код що зчитує інформацію з апаратного токена та комілюється в бібліотеку USB Device Info Extractor.dll)

```
#include "pch.h"
#include <string>
BOOL APIENTRY DllMain(HMODULE hModule,
    DWORD  ul_reason_for_call,
    LPVOID lpReserved
)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
#include <com_serhiihurin_passwordmanager_connector_DllConnector.h>
#include <cstdio>
#include <Windows.h>
#include <iostream>
#include <comdef.h>
#include <Wbemidl.h>
#pragma comment(lib, "wbemuuid.lib")
std::string GetRemovableDriveInfo() {
    HRESULT hres;
    hres = CoInitializeEx(NULL, COINIT_APARTMENTTHREADED);
    if (FAILED(hres)) {
        return "Unable to initialize COM library.";
    }
    hres     =     CoInitializeSecurity(NULL,     -1,     NULL,     NULL,     RPC_C_AUTHN_LEVEL_DEFAULT,
RPC_C_IMP_LEVEL_IMPERSONATE, NULL, EOAC_NONE, NULL
);
    if (FAILED(hres) && hres != RPC_E_TOO_LATE) {
        CoUninitialize(); // Uninitialize COM if it was initialized in this function
        return "Unable to initialize WMI security.";
    }
    IWbemLocator* pLoc = NULL;
    hres = CoCreateInstance(
        CLSID_WbemLocator,
        0,
        CLSCTX_INPROC_SERVER,
        IID_IWbemLocator, (LPVOID*)&pLoc
    );
    if (FAILED(hres)) {
        CoUninitialize(); // Uninitialize COM if it was initialized in this function
        return "Unable to create WbemLocator.";
    }
    IWbemServices* pSvc = NULL;
    hres = pLoc->ConnectServer(_bstr_t(L"ROOT\\CIMV2"), NULL, NULL, 0, NULL, 0, 0, &pSvc);
    if (FAILED(hres)) {
        pLoc->Release();
        CoUninitialize(); // Uninitialize COM if it was initialized in this function
        return "Unable to connect to the WMI service.";
    }
    IEnumWbemClassObject* pEnumerator = NULL;
    hres = pSvc->ExecQuery(
        bstr_t("WQL"),
        bstr_t("SELECT * FROM Win32_DiskDrive WHERE MediaType = 'Removable Media'"),
        WBEM_FLAG_FORWARD_ONLY | WBEM_FLAG_RETURN_IMMEDIATELY, NULL, &pEnumerator);
    if (FAILED(hres)) {
        pSvc->Release();
        pLoc->Release();
        CoUninitialize(); // Uninitialize COM if it was initialized in this function
        return "Request to WMI failed.";
    }
    std::string result;
    IWbemClassObject* pclsObj = NULL;
    ULONG uReturn = 0;
```

```
    while (pEnumerator) {
        HRESULT hr = pEnumerator->Next(WBEM_INFINITE, 1, &pclsObj, &uReturn);
        if (0 == uReturn) {
            break;
        }
        VARIANT vtProp;
        hr = pclsObj->Get(L"DeviceID", 0, &vtProp, 0, 0);
        result += "DeviceID: " + std::string(_bstr_t(vtProp.bstrVal)) + "\n";
        VariantClear(&vtProp);
        hr = pclsObj->Get(L"Model", 0, &vtProp, 0, 0);
        result += "Model: " + std::string(_bstr_t(vtProp.bstrVal)) + "\n";
        VariantClear(&vtProp);
        hr = pclsObj->Get(L"SerialNumber", 0, &vtProp, 0, 0);
        result += "SerialNumber: " + std::string(_bstr_t(vtProp.bstrVal)) + "\n";
        VariantClear(&vtProp);
        pclsObj->Release();
    }
    pSvc->Release();
    pLoc->Release();
    if (pEnumerator)
        pEnumerator->Release();
    CoUninitialize(); // Uninitialize COM
    return result;
}
extern "C" {
    JNIEXPORT                                       jstring                                         JNICALL
Java_com_serhiihurin_passwordmanager_connector_DllConnector_USBInfoExtractor(JNIEnv* env, jobject obj) {
        // Call the function to get information about USB devices and return this information as a string
        std::string info = GetRemovableDriveInfo();
        return env->NewStringUTF(info.c_str());
    }
}
```

DllConnector.java (Клас що підключає .dll бібліотеку до java проекту)

```
package com.serhiihurin.passwordmanager.connector;
public class DllConnector {
    public native String USBInfoExtractor();
    static {
        System.loadLibrary("USB Device Info Extractor");
    }
}
```

USBFlashDriveInfoRetrievalServiceImpl.java (Клас що використовує підключену до проекту .dll бібліотеку та форматує отриману інформацію)

```
package com.serhiihurin.passwordmanager.service;
import com.serhiihurin.passwordmanager.connector.DllConnector;
import com.serhiihurin.passwordmanager.service.interfaces.USBFlashDriveInfoRetrievalService;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import java.util.Objects;
@Service
@RequiredArgsConstructor
public class USBFlashDriveInfoRetrievalServiceImpl implements USBFlashDriveInfoRetrievalService {
    private final DllConnector dllConnector;
    @Override
    public String getUSBFlashDriveInfo() {
        String rawUSBFlashDriveInfo = dllConnector.USBInfoExtractor();
        if (!(Objects.equals(rawUSBFlashDriveInfo, ""))) {
            String[] USBInfoArray = rawUSBFlashDriveInfo.replaceAll("DeviceID: ", "|")
                    .replaceAll("Model: ", "|")
                    .replaceAll("SerialNumber: ", "|")
                    .replaceAll("\n", "")
                    .split("\\|");
            String[] filteredInfoArray = {USBInfoArray[2], USBInfoArray[3]};
            return filteredInfoArray[0] + " " + filteredInfoArray[1];
        } else {
            return rawUSBFlashDriveInfo;
        }
    }
}
```

USBFlashDriveInfoRetrievalFacadeImpl.java (Клас що виконує прив’язку та відв’якзу апаратного токена та формує майстер-пароль)

```
package com.serhiihurin.passwordmanager.facade;
import com.serhiihurin.passwordmanager.entity.Record;
import com.serhiihurin.passwordmanager.entity.User;
```

```
import com.serhiihurin.passwordmanager.enums.ExistingRecordsOperationType;
import com.serhiihurin.passwordmanager.facade.interfaces.USBFlashDriveInfoRetrievalFacade;
import com.serhiihurin.passwordmanager.service.interfaces.EncryptionService;
import com.serhiihurin.passwordmanager.service.interfaces.FileService;
import com.serhiihurin.passwordmanager.service.interfaces.RecordService;
import com.serhiihurin.passwordmanager.service.interfaces.USBFlashDriveInfoRetrievalService;
import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import net.samuelcampos.usbdrivedetector.USBDeviceDetectorManager;
import org.springframework.stereotype.Component;
import java.util.List;
import java.util.Objects;
import java.util.Optional;
import java.util.concurrent.Callable;
@Component
@RequiredArgsConstructor
@Slf4j
public class USBFlashDriveInfoRetrievalFacadeImpl implements USBFlashDriveInfoRetrievalFacade {
    private final USBFlashDriveInfoRetrievalService usbService;
    private final USBDeviceDetectorManager usbDeviceDetectorManager;
    private final FileService fileService;
    private final EncryptionService encryptionService;
    private final RecordService recordService;
    @Override
    public String linkUSBFlashDrive(boolean isRegisterProcess) {
        Callable<String> usbFlashDriveTask = () -> {
            String USBFlashDriveInfo;
            String usbUUID;
            String USBKey;
            do {
                USBFlashDriveInfo = usbService.getUSBFlashDriveInfo();
                usbUUID = usbDeviceDetectorManager.getRemovableDevices().get(0).getUuid();
                if (isRegisterProcess) {
                    fileService.generateKeyFile();
                }
                USBKey = fileService.getKeyFromFile();
                if (!Objects.equals(USBFlashDriveInfo, "")) {
                    log.info("Got string: {}", USBFlashDriveInfo);
                    log.info(usbDeviceDetectorManager.getRemovableDevices().get(0).toString());
                    return USBFlashDriveInfo + usbUUID + USBKey;
                } else {
                    try {
                        log.info("Empty data");
                        Thread.sleep(2000);
                    } catch (InterruptedException e) {
                        log.error(e.getMessage());
                    }
                }
            } while (Objects.equals(USBFlashDriveInfo, ""));
            return null;
        };
        String USBFlashDriveInfo = null;
        try {
            USBFlashDriveInfo = usbFlashDriveTask.call();
        } catch (Exception e) {
            log.error(e.getMessage());
        }
        return USBFlashDriveInfo;
    }
    @Override
    @Transactional
    public void changeUSBFlashDrive(User currentUser, ExistingRecordsOperationType operationType) {
        List<Record> recordList = recordService.getAllRecordsByUserId(currentUser.getUserId());
        for(Record record : recordList) {
            if (operationType == ExistingRecordsOperationType.DECRYPT) {
                record.setUsername(encryptionService.decrypt(record.getUsername(),
Optional.of(currentUser)));
                record.setPassword(encryptionService.decrypt(record.getPassword(),
Optional.of(currentUser)));
            }
            if (operationType == ExistingRecordsOperationType.ENCRYPT) {
                record.setUsername(encryptionService.encrypt(record.getUsername(),
Optional.of(currentUser)));
                record.setPassword(encryptionService.encrypt(record.getPassword(),
Optional.of(currentUser)));
            }
            recordService.updateRecord(record);
        }
```

```
    }
}
```

EncryptionServiceImpl.java (Клас шифрування та розшифрування даних)

```
package com.serhiihurin.passwordmanager.service;
import com.serhiihurin.passwordmanager.entity.User;
import com.serhiihurin.passwordmanager.service.interfaces.AuthenticationService;
import com.serhiihurin.passwordmanager.service.interfaces.EncryptionService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;
import javax.crypto.*;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.Base64;
import java.util.Optional;
@Service
@RequiredArgsConstructor
@Slf4j
public class EncryptionServiceImpl implements EncryptionService {
    private final AuthenticationService authenticationService;
    private final String algorithm = "AES/CBC/PKCS5Padding";
    private final IvParameterSpec iv = loadIv();
    @Override
    public String encrypt(String input, Optional<User> anonymousUser) {
        try {
            Cipher cipher = Cipher.getInstance(algorithm);
            if (anonymousUser.isPresent()) {
                cipher.init(Cipher.ENCRYPT_MODE, getKey(anonymousUser.get().getPassword()), iv);
            } else {
                cipher.init(Cipher.ENCRYPT_MODE,
getKey(authenticationService.getAuthenticatedUser().getPassword()), iv);
            }
            byte[] cipherText = cipher.doFinal(input.getBytes());
            return Base64.getEncoder()
                    .encodeToString(cipherText);
        } catch (NoSuchAlgorithmException | NoSuchPaddingException | InvalidKeyException |
                 InvalidAlgorithmParameterException | IllegalBlockSizeException | BadPaddingException e)
{
            throw new RuntimeException(e);
        }
    }
    @Override
    public String decrypt(String cipherText, Optional<User> anonymousUser) {
        try {
            Cipher cipher = Cipher.getInstance(algorithm);
            if (anonymousUser.isPresent()) {
                cipher.init(Cipher.DECRYPT_MODE, getKey(anonymousUser.get().getPassword()), iv);
            } else {
                cipher.init(Cipher.DECRYPT_MODE,
getKey(authenticationService.getAuthenticatedUser().getPassword()), iv);
            }
            byte[] plainText = cipher.doFinal(Base64.getDecoder()
                    .decode(cipherText));
            return new String(plainText);
        } catch (NoSuchAlgorithmException | NoSuchPaddingException | InvalidKeyException |
                 InvalidAlgorithmParameterException | IllegalBlockSizeException | BadPaddingException e)
{
            throw new RuntimeException(e);
        }
    }
    private SecretKey getKey(String input) {
        int midIndex = input.length() / 2;
        String firstHalf = input.substring(0, midIndex);
        String secondHalf = input.substring(midIndex);
        String rearrangedFirstHalf = rearrangeString(firstHalf);
        String rearrangedSecondHalf = rearrangeString(secondHalf);
        String combined = rearrangedSecondHalf + rearrangedFirstHalf ;
        if (combined.length() > 32) {
            combined = combined.substring(0, 32);
        }
        return new SecretKeySpec(combined.getBytes(), "AES");
    }
```

```
    private String rearrangeString(String str) {
        char[] chars = str.toCharArray();
        for (int i = 0; i < chars.length - 1; i += 2) {
            char temp = chars[i];
            chars[i] = chars[i + 1];
            chars[i + 1] = temp;
        }
        return new String(chars);
    }
    private IvParameterSpec loadIv() {
        try {
            byte[] ivBytes = Files.readAllBytes(Paths.get("iv.key"));
            return new IvParameterSpec(ivBytes);
        } catch (IOException e) {
            throw new RuntimeException("Failed to load IV", e);
        }
    }
}
```

GeneratorServiceImpl.java (Клас генерації паролів та ID об’єктів)

```
package com.serhiihurin.passwordmanager.service;
import com.serhiihurin.passwordmanager.enums.EntityType;
import com.serhiihurin.passwordmanager.service.interfaces.GeneratorService;
import com.vaadin.flow.component.notification.Notification;
import org.springframework.stereotype.Service;
import java.security.SecureRandom;
import java.util.Random;
@Service
public class GeneratorServiceImpl implements GeneratorService {
    @Override
    public String generateEntityId(EntityType entityType) {
        String generatedId = "";
        Random random = new Random();
        StringBuilder stringBuilder = new StringBuilder();
        int idLength = 20;
        String characters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
        for (int i = 0; i < idLength; i++) {
            int randomIndex = random.nextInt(characters.length());
            stringBuilder.append(characters.charAt(randomIndex));
        }
        switch (entityType) {
            case USER -> generatedId = "uid_" + stringBuilder;
            case RECORD -> generatedId = "rid_" + stringBuilder;
            case GROUP -> generatedId = "gid_" + stringBuilder;
        }
        if (generatedId.isEmpty()) {
            throw new RuntimeException("Error while generating entity ID");
        }
        return generatedId;
    }
    @Override
    public String generatePassword(
            String length,
            boolean isUpperCaseIncluded,
            boolean isNumbersIncluded,
            boolean isSpecialCharactersIncluded
    ) {
        String UPPER_CASE = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        String LOWER_CASE = "abcdefghijklmnopqrstuvwxyz";
        String NUMBERS = "0123456789";
        String SPECIAL_CHARACTERS = "*$\\|/!@#^?&:;*()-_+=.,'";
        int passwordLength;
        try {
            passwordLength = Integer.parseInt(length);
        } catch (NumberFormatException e) {
            Notification.show("Invalid length");
            return null;
        }
        String generationCharacters = LOWER_CASE;
        if (isUpperCaseIncluded) {
            generationCharacters += UPPER_CASE;
        }
        if (isNumbersIncluded) {
            generationCharacters += NUMBERS;
        }
        if (isSpecialCharactersIncluded) {
            generationCharacters += SPECIAL_CHARACTERS;
        }
        SecureRandom random = new SecureRandom();
```

```
        StringBuilder generatedPassword = new StringBuilder();
        for (int i = 0; i < passwordLength; i++) {
            generatedPassword.append(
                    generationCharacters.charAt(
                            random.nextInt(generationCharacters.length())
                    )
            );
        }
        return generatedPassword.toString();
    }
}
```

SecurityConfig.java (Клас налаштувань безпеки веб-додатку)

```
package com.serhiihurin.passwordmanager.config;
import com.serhiihurin.passwordmanager.filter.JWTAuthenticationFilter;
import com.serhiihurin.passwordmanager.filter.UserLogInSuccessHandler;
import com.serhiihurin.passwordmanager.views.list.LoginView;
import com.vaadin.flow.spring.security.VaadinWebSecurity;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.oauth2.jose.jws.JwsAlgorithms;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig extends VaadinWebSecurity {
    private final UserLogInSuccessHandler userLogInSuccessHandler;
    private final AuthenticationProvider authenticationProvider;
    private final JWTAuthenticationFilter jwtAuthenticationFilter;
    @Value("${application.security.jwt.secret-key}")
    private String authSecret;
    @Value("${application.security.jwt.expiration}")
    private long jwtExpiration;
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .formLogin(
                        form -> form
                                .loginPage("/login")
                                .successHandler(userLogInSuccessHandler)
                                .failureUrl("/login?error=true")
                )
                .csrf(AbstractHttpConfigurer::disable)
                .cors(AbstractHttpConfigurer::disable)
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers(
                                AntPathRequestMatcher.antMatcher(HttpMethod.GET, "/images/*.png")
                        ).permitAll()
                        .requestMatchers(
                                new AntPathRequestMatcher("/line-awesome/**/*.svg")
                        ).permitAll()
                )
                .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);
        super.configure(http);
        setLoginView(http, LoginView.class);
        setStatelessAuthentication(
                http,
                new SecretKeySpec(Base64.getDecoder().decode(authSecret), JwsAlgorithms.HS256),
                "com.serhiihurin.passwordmanager",
                jwtExpiration
        );
    }
}
```

JWTAuthenticationFilter.java (Клас обробки jwt-токена що формується при автентифікації та визначає тривалість сесії)

```
package com.serhiihurin.passwordmanager.filter;
import com.serhiihurin.passwordmanager.service.interfaces.JWTService;
import jakarta.servlet.*;
import lombok.RequiredArgsConstructor;
```

```
import org.springframework.lang.NonNull;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.authentication.AuthenticationFailureHandler;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import java.io.IOException;
@Component @RequiredArgsConstructor
public class JWTAuthenticationFilter extends OncePerRequestFilter {
    private final JWTService jwtService; private final AuthenticationFailureHandler failureHandler;
    protected void doFilterInternal(@NonNull HttpServletRequest request, @NonNull HttpServletResponse
response, @NonNull FilterChain filterChain) throws ServletException, IOException {
        try {
            String token = jwtService.extractToken(request);
            if (token != null && jwtService.isTokenValid(token)) {throw new AuthenticationException("Token
expired") {};}
            filterChain.doFilter(request, response);
        } catch (AuthenticationException e) {
            SecurityContextHolder.clearContext();        failureHandler.onAuthenticationFailure(request,
response, e);}
    }
}
```

73

Додаток В

ІЛЮСТРАТИВНА ЧАСТИНА

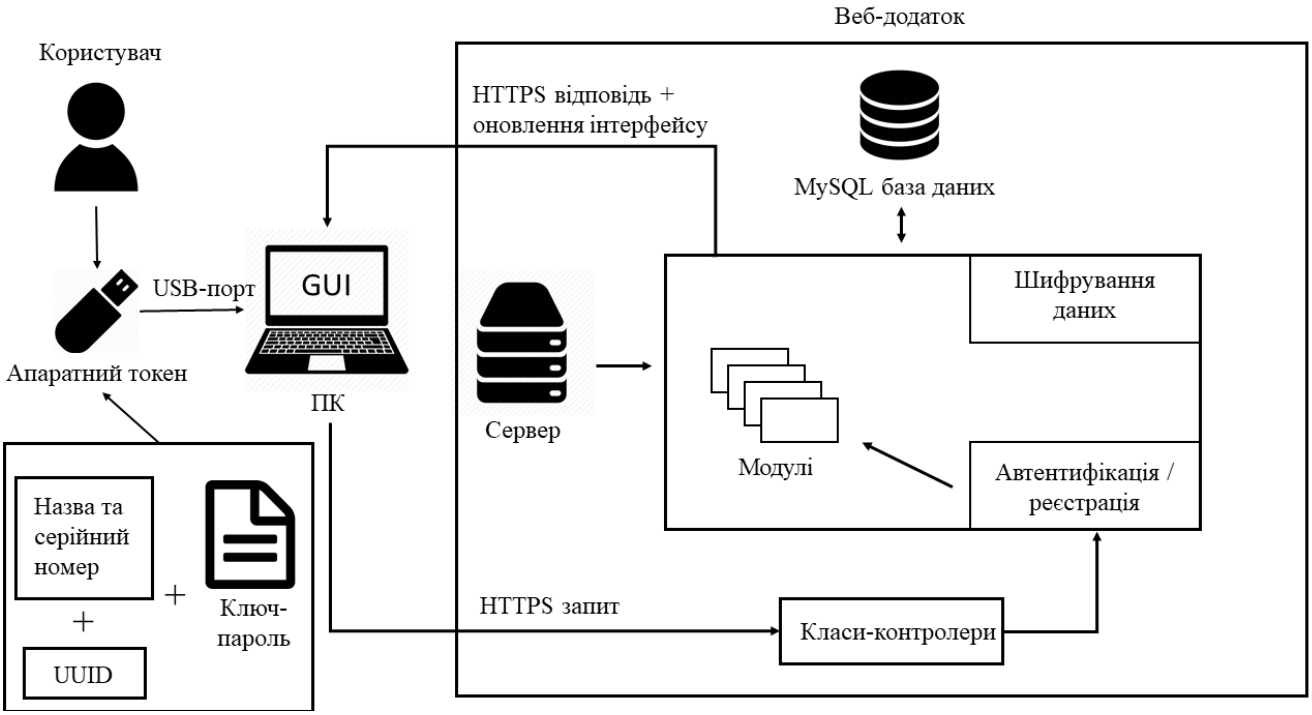
ЗАСІБ ДЛЯ КЕРУВАННЯ ПАРОЛЯМИ НА ОСНОВІ АПАРАТНОГО ТОКЕНА

Виконав: студент 4 курсу групи 1БС-20б
спеціальності 125 Кібербезпека
_____Сергій ГУРІН
14 червня 2024 р.

Керівник: к. т. н., доц., доцент каф. ЗІ
_____Олеся ВОЙТОВИЧ
14 червня 2024 р.

3.6

ЗАГАЛЬНА СТРУКТУРА ПРОГРАМНОГО ЗАСОБУ



ЗАГАЛЬНА СТРУКТУРА РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАСОБУ ТА ЗВ’ЯЗКІВ МІЖ КЛАСАМИ

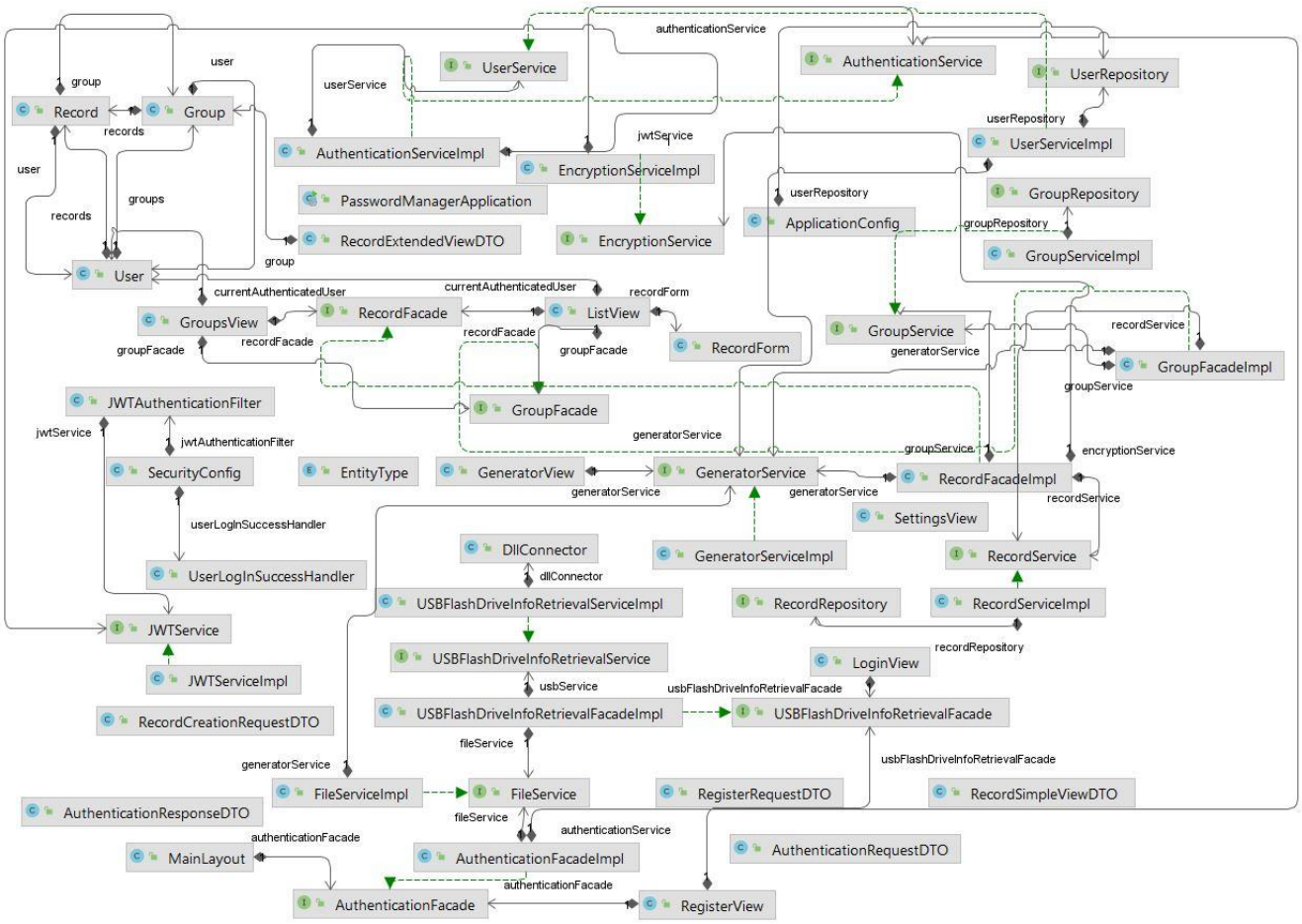


СХЕМА РОБОТИ АЛГОРИТМУ ЗЧИТУВАННЯ ІНФОРМАЦІЇ З АПАРАТНОГО ТОКЕНУ

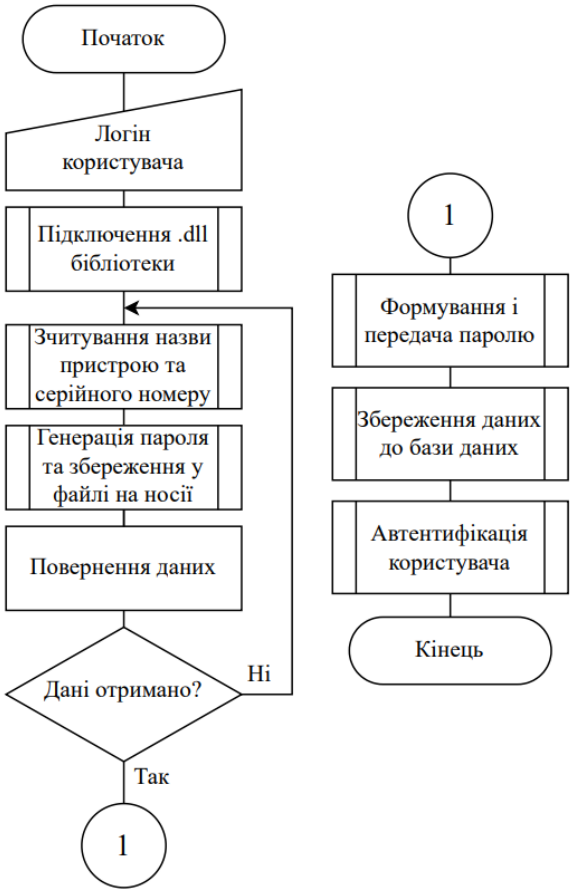


СХЕМА РОБОТИ АЛГОРИТМУ АВТЕНТИФІКАЦІЇ КОРИСТУВАЧА

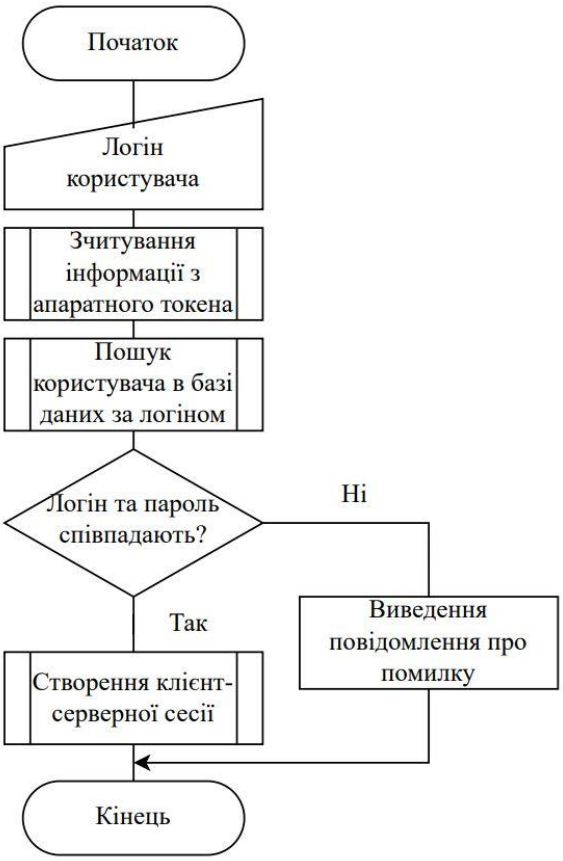


СХЕМА ПЕРЕДАЧІ ЗАПИТІВ НА ВИКОНАННЯ ОПЕРАЦІЙ ЧИТАННЯ, ЗБЕРЕЖЕННЯ, ОНОВЛЕННЯ ТА ВИДАЛЕННЯ ЗАПИСІВ ДО БАЗИ ДАНИХ

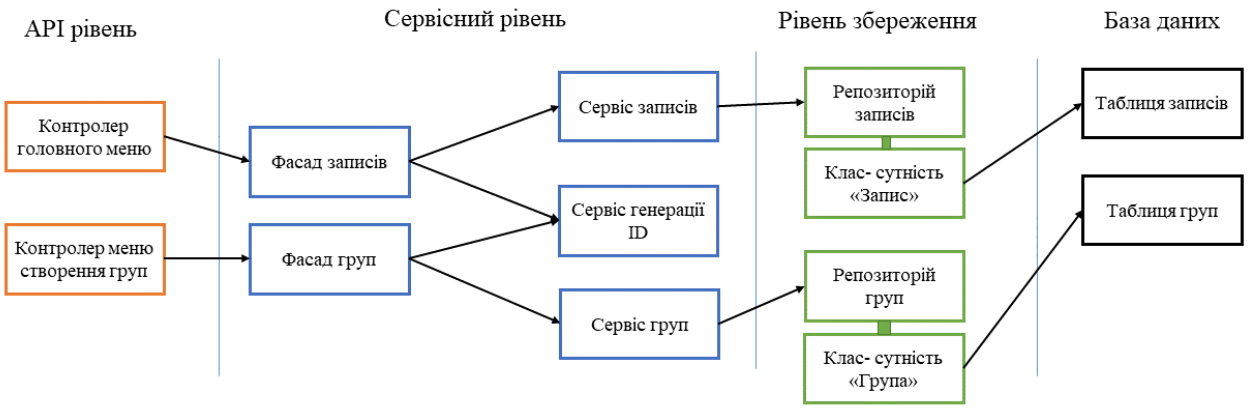


СХЕМА РОБОТИ АЛГОРИТМУ ГЕНЕРАЦІЇ ПАРОЛІВ

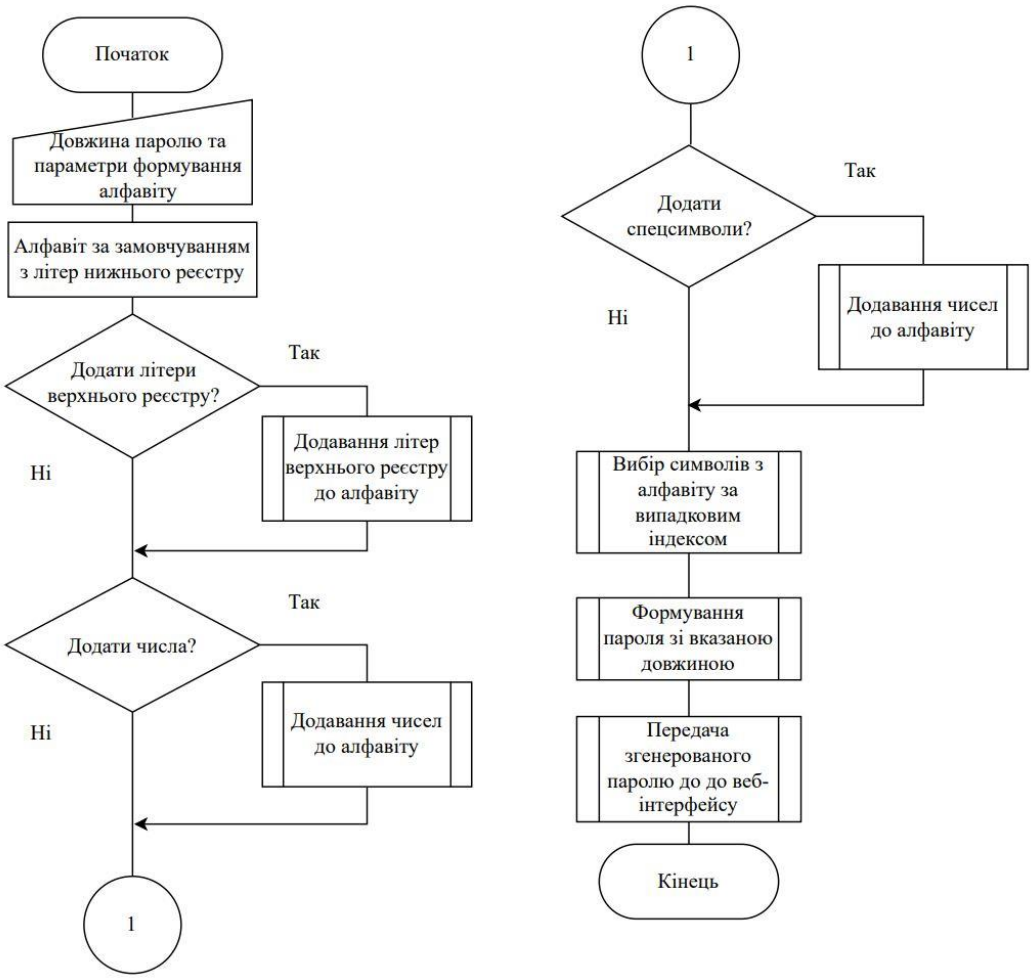
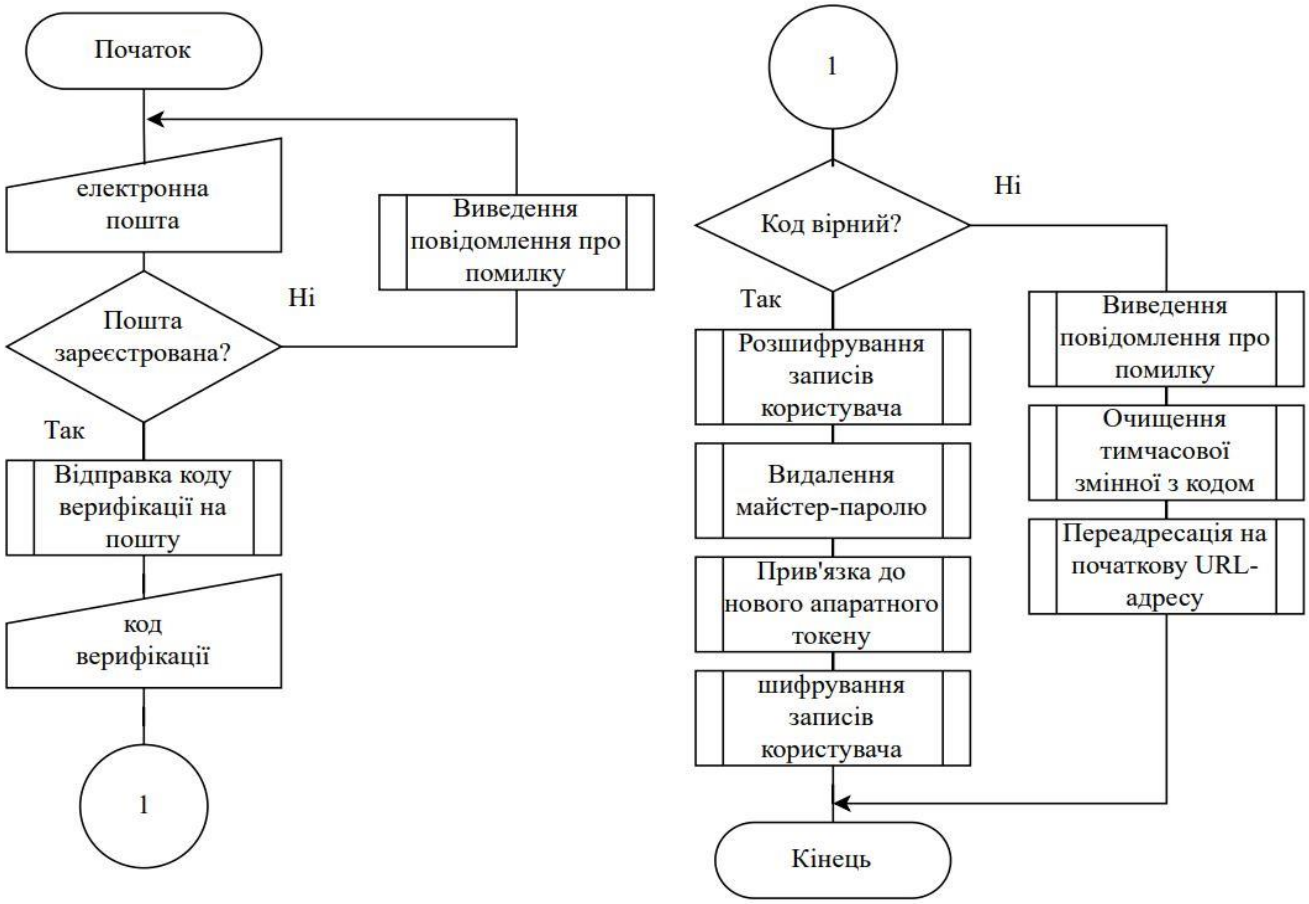


СХЕМА АЛГОРИТМУ ГЕНЕРАЦІЇ СЕКРЕТНОГО КЛЮЧА ШИФРУВАННЯ

СХЕМА РОБОТИ АЛГОРИТМУ ОНОВЛЕННЯ МАЙСТЕР-ПАРОЛЮ ТА ПРИВ’ЯЗКИ ДО НОВОГО АПАРАТНОГО ТОКЕНА



РЕЗУЛЬТАТИ ТЕСТУВАННЯ