Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти
(повна назва)

Кафедра програмної інженерії
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Програмна система шифрування на основі біометричних даних
(тема)

Виконав:
студент 2 курсу, групи ПЗПІп-22-2

Гавриленко О.С.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник ст.викл. Олійник О.О.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

**ЗАВДАННЯ**
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентці Гавриленко Олені Сергіївні
(прізвище, ім'я, по батькові)

1. Тема роботи Програмна система шифрування на основі біометричних даних
Затверджена наказом по університету від 17.06. 2024р. № 588 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 23.07.2024 р.
3. Вихідні дані до роботи Розробити програмну систему, здатну шифрувати та розшифровувати дані, формуючи ключі шифрування на основі біометричних даних людини.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі та постановка задачі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

# КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Термін | Відмітки про виконання |
|---|---|---|---|
| 1 | Аналіз предметної галузі, огляд існуючих рішень вибір найбільш придатних аналогів | 18.06.2024 | Виконано |
| 2 | Формування вимог до ПЗ | 20.06.2024 | Виконано |
| 3 | Розробка архітектури ПЗ та UML-проектування | 27.06.2024 | Виконано |
| 4 | Розробка та тестування програмного забезпечення | 05.07.2024 | Виконано |
| 5 | Написання пояснювальної записки | 10.07.2024 | Виконано |
| 6 | Перевірка пояснювальної записки керівником, підготовка роботи для проходження перевірки на антиплагіаті та проходження норм контролю | 13.07.2024 | Виконано |
| 7 | Отримання відзиву від керівника кваліфікаційної роботи, попередній захист роботи та проходження нормо контролю, оцінка роботи рецензентом. | 16.07.2024 | |
| 8 | Здача роботи в електронний архів, допуск роботи до захисту завідувачем кафедри та надання готової роботи секретарю ЕК | 20.07.2024 | |
| 9 | Захист кваліфікаційної роботи | 23.07.2024 | |

Дата видачі завдання 17 06 2024р.

Студент (ка) _____ Гавриленко О.С.
(підпис)

Керівник роботи _____ ст.викл. Олійник О.О.
(підпис)

# РЕФЕРАТ / ABSTRACT

Звіт з передатестаційної практики: 72 с., 54 рис., 5 таблиць, 15 джерел.

БІОМЕТРИЧНІ ДАНІ, ШИФРУВАННЯ, RSA КЛЮЧІ, WEB-ДОДАТОК, FACE-API, H2, HIBERNATE, JAVA, JAVASCRIPT, SPRING.

Об’єктом розробки є програмна система шифрування на основі біометричних даних обличчя. Ця система призначена для забезпечення високого рівня безпеки конфіденційних даних шляхом використання унікальних фізіологічних характеристик користувача. Розробка включає клієнтську та серверну частини, що працюють у тісній взаємодії для динамічного формування криптографічних ключів і захисту інформації.

Об’єктом дослідження є процеси зчитування та обробки біометричних даних обличчя, а також методи використання цих даних для генерації криптографічних ключів. Дослідження охоплює аналіз точності та стабільності біометричних даних, а також ефективність різних алгоритмів стабілізації зображень і фільтрації шумів. Особлива увага приділяється порівнянню існуючих методів біометричної аутентифікації та їхньої інтеграції з сучасними криптографічними техніками.

Для вирішення завдання використовуються сучасні технології та бібліотеки, такі як JavaScript з face-api.js для клієнтської частини та Java з Spring Boot і Hibernate для серверної частини. Основні методи включають зчитування біометричних даних обличчя за допомогою камери, обробку цих даних для виділення ключових характеристик, та динамічне формування RSA ключів на основі отриманих біометричних відбитків. Для забезпечення точності та стабільності використовуються алгоритми стабілізації зображень і фільтрації шумів, а також збирання метрик протягом певного часу для вибору стабільних значень.

Результатом роботи є розроблена програмна система шифрування на основі біометричних даних обличчя, що забезпечує високий рівень безпеки

конфіденційних даних. Система включає ефективні методи зчитування та обробки біометричних даних, динамічне формування криптографічних ключів та інтеграцію з базою даних H2 для зберігання симетричних ключів. Тестування підтвердило стабільність і надійність системи, що дозволяє рекомендувати її для використання в різних сферах, де необхідний високий рівень захисту інформації.

WEB APPLICATION, BIOMETRIC DATA, RSA KEYS, ENCRYPTION, JAVASCRIPT, FACE-API, JAVA, SPRING, HIBERNATE, H2.

The object of development is a software system for encryption based on facial biometric data. This system is designed to ensure a high level of security for confidential data by utilizing the unique physiological characteristics of the user. The development includes both client-side and server-side components that work in close interaction to dynamically generate cryptographic keys and protect information.

The object of research includes the processes of capturing and processing facial biometric data, as well as the methods of using this data to generate cryptographic keys. The research covers the analysis of accuracy and stability of biometric data, as well as the efficiency of various algorithms for image stabilization and noise filtering. Special attention is given to comparing existing methods of biometric authentication and their integration with modern cryptographic techniques.

To solve the task, modern technologies and libraries are used, such as JavaScript with face-api.js for the client-side and Java with Spring Boot and Hibernate for the server-side. The main methods include capturing facial biometric data using a camera, processing this data to extract key characteristics, and dynamically generating RSA keys based on the obtained biometric fingerprints. Algorithms for image stabilization and noise filtering are employed to ensure accuracy and stability, along with collecting metrics over a period of time to select stable values.

The result of the work is a developed software system for encryption based on facial biometric data, ensuring a high level of security for confidential data. The system includes efficient methods for capturing and processing biometric data, dynamic

generation of cryptographic keys, and integration with the H2 database for storing symetric keys. Testing confirmed the stability and reliability of the system, allowing it to be recommended for use in various fields where a high level of information security is required.

Я, Гавриленко Олена Сергіївна, студентка гр. ПЗПІп-22-2, здобувач вищої освіти на першому (бакалаврському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Програмна система шифрування на основі біометричних даних», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі запозичення з друкованих та електронних джерел мають відповідні посилання. Я ознайомлена з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови до допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

# ЗМІСТ

# ВСТУП

У сучасному світі інформаційна безпека набуває все більшого значення, оскільки обсяги конфіденційних даних стрімко зростають, а технологічні можливості зловмисників стають дедалі складнішими. Однією з найперспективніших технологій у сфері захисту інформації є біометричні методи аутентифікації та шифрування даних. Вони базуються на унікальних фізіологічних характеристиках людини, що ускладнює підробку або крадіжку даних.

В рамках виконання даної роботи було розроблено програмну систему шифрування на основі біометричних даних обличчя, що поєднує в собі технології в області криптографії та біометрії. Головна мета дослідження полягає у створенні ефективної та надійної системи, яка динамічно генеруватиме криптографічні ключі на основі зчитаних біометричних даних, забезпечуючи високий рівень захисту інформації.

У даній роботі розглядаються основні принципи біометричних систем аутентифікації, аналізуються сучасні методи шифрування та можливості їхнього застосування для захисту конфіденційних даних. Особлива увага приділяється проблемам стабільності та точності зчитування біометричних даних, а також методам їхнього вирішення.

Проектування системи охоплює як клієнтську, так і серверну частини, де використання сучасних технологій, таких як JavaScript, face-api.js, Java, Spring Boot та Hibernate, забезпечує високу продуктивність, масштабованість та надійність. База даних H2 використовується для зберігання симетричних ключів, що дозволяє забезпечити швидкий доступ до них та легкість адміністрування.

Таким чином, розроблена система спрямована на підвищення рівня безпеки в інформаційному середовищі, забезпечуючи надійний захист конфіденційних даних за допомогою інноваційних біометричних та криптографічних методів.

# 1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

## 1.1 Аналіз предметної галузі

Захист інформації є однією з ключових задач сучасного світу. З розвитком інформаційних технологій та зростанням обсягів передаваних і збережених даних, питання кібербезпеки стають все більш актуальними. В умовах швидкого розвитку технологій, інтернету речей (IoT), хмарних обчислень та мобільних платформ, захист даних від несанкціонованого доступу та забезпечення конфіденційності є першочерговими завданнями для багатьох організацій та приватних користувачів.

Використання традиційних методів аутентифікації, таких як паролі та PIN-коди, має ряд недоліків. По-перше, вони схильні до атак методом перебору та фішингових атак, що може призвести до компрометації облікових даних користувачів. По-друге, користувачі часто використовують слабкі або однакові паролі для різних систем, що підвищує ризик зломів. Крім того, необхідність запам’ятовування численних паролів створює додаткове навантаження на користувачів, що може призвести до втрати або неправильного використання паролів.

Біометричні методи аутентифікації, такі як відбитки пальців, розпізнавання обличчя, райдужна оболонка ока, голос та інші фізіологічні характеристики, пропонують більш високий рівень безпеки, оскільки вони важко підробити або вкрасти. Біометричні дані є унікальними для кожної людини і забезпечують більш надійний захист від несанкціонованого доступу. Використання біометричних даних також спрощує процес аутентифікації, оскільки користувачам не потрібно запам’ятовувати складні паролі.

Проте, самостійне використання біометричних даних для аутентифікації має свої обмеження. Виникають проблеми з точністю зчитування даних в умовах змінного освітлення, різних кутів нахилу обличчя, змін у зовнішності користувача (наприклад, через зміну зачіски або носіння окулярів). Також виникають питання захисту самих біометричних даних від несанкціонованого доступу та можливості їх підробки.[6]

Використання біометричних даних для шифрування інформації є інноваційним підходом, що дозволяє поєднати переваги обох методів - шифрування та біометричної аутентифікації. Такий підхід дозволяє створити унікальний криптографічний ключ для кожного користувача на основі його біометричних даних, що значно підвищує рівень безпеки та знижує ризики несанкціонованого доступу. Важливим аспектом є те, що біометричні дані не зберігаються у вигляді звичайних файлів, а використовуються для динамічного генерування ключів при кожному зчитуванні.

Технологія шифрування на основі біометричних даних може знайти широке застосування в різних галузях: від забезпечення безпеки мобільних пристроїв і персональних комп'ютерів до захисту конфіденційної інформації в корпоративних системах та банківській сфері. Вона також може бути використана для забезпечення безпеки в інтернеті речей, де необхідно забезпечити захист великої кількості пристроїв та даних, що передаються між ними.[1]

Таким чином, розробка системи шифрування на основі біометричних даних є актуальною та своєчасною задачею, яка може значно підвищити рівень безпеки інформації та забезпечити надійний захист від несанкціонованого доступу. Враховуючи зростаючі загрози кібербезпеки та необхідність захисту конфіденційної інформації, розробка таких систем має важливе значення для сучасного суспільства та технологічного розвитку.

## 1.2 Виявлення проблем та актуалізація рішень

Основною проблемою, з якою стикаються системи, що використовують біометричні дані для шифрування, є необхідність забезпечення стабільності та точності зчитуваних даних. Відхилення в умовах зчитування, таких як освітлення, кут нахилу обличчя, зміни у зовнішньому вигляді користувача, можуть призводити до похибок і ускладнювати процес генерації стабільних криптографічних ключів. Ця проблема є особливо актуальною для систем, які використовують динамічно сформовані ключі, оскільки навіть невеликі відхилення можуть призвести до неможливості дешифрування даних.

Основні проблеми, які виникають при розпізнаванні обличчя:

– освітлення та кут нахилу обличчя: зміна освітлення або кута нахилу обличчя може впливати на якість зображення і, відповідно, на точність зчитування біометричних даних. Наприклад, сильне освітлення може створювати відблиски, які спотворюють зображення, а тіні можуть робити деякі частини обличчя менш видимими. Зміни в куті нахилу обличчя можуть змінювати відстань між ключовими точками, що також впливає на результати зчитування;
– зміни у зовнішньому вигляді користувача: зміна зачіски, носіння окулярів або інших аксесуарів, наявність або відсутність бороди можуть впливати на точність зчитування біометричних даних. Ці зміни можуть призводити до значних відхилень у зчитуваних метриках, що робить важким формування стабільного біометричного відбитка;
– нестабільність даних: біометричні дані, такі як відстань між очима або форма носа, можуть змінюватися навіть при невеликих рухах користувача або через різні умови зчитування. Це може призводити до нестабільності зчитуваних даних, що робить складним використання цих даних для формування криптографічних ключів.[2]

Для досягнення поставлених цілей та для вирішення проблеми стабільності зчитуваних даних у даному дослідженні використовувалися наступні методи:

– аналіз літературних джерел: проведено огляд наукової та технічної літератури, статей, досліджень, що стосуються шифрування, біометричної аутентифікації та використання біометричних даних в системах безпеки;
– експериментальні дослідження: виконано серію експериментів зі зчитування та обробки біометричних даних за допомогою різних методів та технологій, включаючи використання бібліотеки face-api.js. Під час реалізації першого та другого етапів з’ясувалося, що необхідно мінімізувати похибки біометричних даних при кожному зчитуванні. Оскільки система не зберігає приватний ключ, а формує його за заданим

алгоритмом при кожному зчитуванні біометричних даних, виникла необхідність забезпечити статичність отриманих даних незалежно від кута нахилу обличчя, зачіски, освітлення тощо. Проведено експерименти з різними методами зчитування, включаючи зчитування пропорції розміру трапеції (відстань між очима та кутами губ) до розміру обличчя, але виявилося, що ці розміри дуже схожі навіть у людей з не схожими обличчями. Тоді вирішено використовувати пропорцію замірів очей до всього обличчя. Хоча при використанні такого методу програма видає однакові результати для схожих облич, це є допустимим на даному етапі розробки та досліджень;

– консультації зі спеціалістами: було проведено консультації з експертами зі штучного інтелекту та лікарями для мінімізації похибок при зчитуванні біометричних даних, аналізу результатів та визначення найефективніших підходів. Під час консультацій з'ясувалося, що в 100 відсотках випадків неможливо отримувати однакові біометричні дані з обличчя людини, в тому числі через малі розміри пікселів, у яких виконуються заміри;
– розробка програмного забезпечення: використано мову програмування JavaScript та бібліотеку face-api.js для реалізації алгоритмів зчитування біометричних даних та формування криптографічних ключів. Для формування пари RSA ключів вирішено використовувати два числа: перше число формується на основі біометричних даних, а друге число – на основі матриці символів, яка може бути індивідуальною для кожного екземпляра системи. Символи в матриці обираються на основі числа, отриманого з біометричних даних;
– тестування: проведено тестування розробленої системи в різних умовах, зокрема при різних кутах нахилу обличчя, змінному освітленні та інших змінних факторах, для оцінки її ефективності та стабільності. Виявилося, що система не завжди отримує однакове число при кожному зчитуванні, проте часто отримує стале число зі зчитування біометрії одного обличчя.

В результаті було прийнято рішення збирати протягом 30 секунд метрики та обирати число, яке найчастіше зустрічається. Це число використовується для побудови ключів шифрування.

## 1.3 Аналіз аналогів програмного забезпечення

Існує кілька відомих програмних продуктів, що використовують біометричні дані для шифрування та аутентифікації. Однак, кожен з них має свої особливості, переваги та недоліки. Розглянемо деякі з найбільш популярних аналогів програмного забезпечення для шифрування з використанням біометричних даних (див. таблицю 1.1).

Таблиця 1.1 – Порівняльний аналіз аналогів програмного забезпечення

| Опис | Переваги | Недоліки |
|---|---|---|
| Microsoft Windows Hello | | |
| Windows Hello є вбудованою функцією аутентифікації в операційній системі Microsoft Windows, яка використовує біометричні дані, такі як розпізнавання обличчя, відбитки пальців і райдужну оболонку ока, для швидкого та безпечного входу в систему. | – глибока інтеграція з операційною системою Windows забезпечує безперебійний досвід користувача;<br>– підтримка різних біометричних методів дозволяє користувачам вибирати найбільш зручний метод аутентифікації;<br>– використання сучасних алгоритмів забезпечує високу точність розпізнавання. | – вразливість до зовнішніх факторів: (освітлення, кут нахилу обличчя та інші фактори можуть впливати на точність розпізнавання);<br>– вимоги до апаратного забезпечення (потребує спеціальних сенсорів та камер, які не завжди є доступними на всіх пристроях). |

Кінець таблиці 1.1

| Apple Face ID | | |
|---|---|---|
| Face ID – це технологія розпізнавання обличчя, розроблена Apple, яка використовується в їхніх мобільних пристроях для розблокування телефону, аутентифікації в додатках та здійснення платежів. | – використання технології глибини та інфрачервоних датчиків забезпечує високу точність та надійність;<br>– працює з різними додатками та службами Apple;<br>– користувачам не потрібно торкатися пристрою, що підвищує гігієнічність та зручність. | – може важко розпізнати обличчя під певними кутами чи відстанями;<br>– технологія вимагає використання дорогих компонентів, що може підвищувати вартість пристроїв. |
| Дія.Підпис | | |
| Дія.Підпис – це електронний підпис, що є частиною державного додатку “Дія”. Ця технологія дозволяє користувачам підписувати документи та здійснювати транзакції онлайн, використовуючи біометричні дані, зокрема розпізнавання обличчя. | – дозволяє користувачам швидко та безпечно підписувати документи онлайн без необхідності фізичної присутності;<br>– використовує сучасні технології розпізнавання обличчя, що забезпечує надійність та точність аутентифікації. | – вразливість до зовнішніх факторів (освітлення, кут нахилу обличчя та інші фактори можуть впливати на точність розпізнавання);<br>– працює лише в межах України та з українськими державними сервісами. |

Аналіз аналогів програмного забезпечення для шифрування з використанням біометричних даних показує, що існуючі рішення мають свої переваги та недоліки. Найбільш розповсюдженими методами є розпізнавання обличчя та відбитків пальців, завдяки їх зручності та точності. Однак, кожен метод має свої обмеження, пов’язані з зовнішніми факторами та вартістю обладнання.

Програмні рішення, такі як Microsoft Windows Hello та Apple Face ID, демонструють високу інтеграцію з операційними системами та екосистемами виробників, що забезпечує безшовний досвід користувача. Проте, навіть їх ефективність може знижуватись під впливом зовнішніх факторів.

Однак, основною відмінністю розроблюваної нами системи від проаналізованих аналогів є те, що розроблена нами система не зберігає персональних даних. Натомість, аналогічні системи працюють за принципом коли вони зберігають біометричну інформацію та потім при зчитуванні порівнюють отримані дані зі збереженими.

## 1.4 Постановка задачі

Ідея програмного продукту полягає у вираженні аналізу та вирішенні проблем, описаних у попередньому розділі. Метою роботи є розробка програмної системи шифрування на основі біометричних даних обличчя. Наша система має вирішувати виділені проблеми цілком чи частково.

Розроблений програмний продукт повинен надавати такі можливості:

– зчитування біометричних даних обличчя користувача;
– обробка зчитаних біометричних даних для виділення ключових характеристик;
– генерація пари RSA ключів на основі біометричних даних обличчя користувача;
– збереження симетричного ключа та динамічна генерація пари RSA ключів при кожному зчитуванні біометричних даних;

- шифрування та дешифрування даних за допомогою згенерованих ключів;
- забезпечення стабільності та точності зчитуваних даних шляхом стабілізації зображення, збирання метрик протягом певного часу та фільтрації шумів.

Програмний продукт має відповідати сучасним стандартам безпеки, бути модульним та гнучким для забезпечення легкості масштабування та підтримки.

Програмна система – це продукт, який забезпечує зчитування, обробку біометричних даних, генерацію криптографічних ключів та шифрування/дешифрування даних. Його модулі мають бути реалізовані таким чином, щоб з мінімальним навантаженням обробляти та надавати дані користувачеві.

Система повинна використовувати сучасні технології розробки програмного забезпечення, а саме задовольняти наступним вимогам:

- мова програмування: JavaScript для клієнтської та Java для серверної частини;
- бібліотеки: face-api.js для зчитування та обробки біометричних даних, Spring Boot, Hibernate для розгортання серверної частини та роботою з базою даних;
- база даних: H2 для збереження симетричних ключів та інших необхідних даних.

Таким чином, розроблена система шифрування на основі біометричних даних обличчя повинна забезпечити високий рівень безпеки та захисту інформації, відповідати сучасним технологічним стандартам та вирішувати виявлені проблеми, описані в попередньому розділі.

# 2 ПЕРЕЛІК ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

## 2.1 Постановка мети

Метою розробки є створення програмної системи шифрування на основі біометричних даних обличчя для забезпечення високого рівня безпеки даних. Ця система повинна динамічно формувати криптографічні ключі на основі унікальних фізіологічних характеристик користувача, що значно підвищить захищеність інформації та знизить ризик несанкціонованого доступу.

Для розробки клієнтської частини буде використовуватися мова програмування JavaScript з бібліотекою face-api.js для зчитування та обробки біометричних даних. Серверна частина буде реалізована на мові Java з використанням Spring Boot та Hibernate для розгортання серверної частини та роботи з базою даних. База даних H2 буде використовуватися для збереження симетричних ключів.

Глобальна архітектура системи – клієнт-серверна. Основні компоненти системи включають модуль зчитування біометричних даних, модуль обробки даних, модуль генерації ключів та модуль шифрування/дешифрування даних.

Серверна частина системи буде розроблена по багаторівневій архітектурі:

– шар стійкості (репозиторії, які відповідають за взаємодію з базою даних);
– шар бізнес-логіки (сервіси, які реалізують основну логіку роботи системи);
– шар презентації (контролери, які забезпечують взаємодію користувачів із системою через API).

Для взаємодії на шарі стійкості буде використовуватися Java-бібліотека Hibernate, яка забезпечує зручну роботу з базою даних. Всі шари будуть залежати від абстракцій, згідно з принципами SOLID (single responsibility, open–closed, Liskov substitution, interface segregation, and dependency inversion). Дана архітектура була обрана через те, що кожен рівень легко передати іншому розробнику (за потреби), легко оновлювати та підтримувати. Також така система і клієнт-серверна архітектура дозволяють легко масштабуватися.

Глобальна архітектура клієнт-сервер була обрана через її актуальність серед бізнес-рішень та здатність надавати зрозумілий і структурований спосіб розробки серверної (бекенд) та клієнтської (фронтенд) частин програмного забезпечення.

## 2.2 Загальний опис системи

Система являє собою клієнт-серверну архітектуру, яка включає модуль зчитування біометричних даних, модуль обробки даних, модуль генерації ключів та модуль шифрування/дешифрування даних. Клієнтська частина реалізована на мові програмування JavaScript з використанням бібліотеки face-api.js для зчитування та обробки біометричних даних. Серверна частина розроблена на мові програмування Java з використанням Spring Boot та Hibernate для розгортання серверної частини та роботи з базою даних. База даних H2 використовується для збереження симетричних ключів.

Система повинна забезпечити високий рівень безпеки та захисту даних шляхом використання біометричних даних обличчя для динамічного формування криптографічних ключів.

## 2.3 Основний функціонал системи

Програмна система шифрування на основі біометричних даних обличчя передбачає реалізацію ряду ключових функцій, що забезпечують її ефективність та надійність.

Однією з основних функцій є зчитування біометричних даних обличчя користувача. Це відбувається за допомогою камери, яка отримує зображення обличчя. Система використовує бібліотеку face-api.js для обробки цих зображень, виділення ключових характеристик обличчя, таких як відстань між очима, форма носа та інші важливі біометричні параметри. Цей етап є критичним, оскільки забезпечує початкові дані для подальших операцій.

Після зчитування біометричних даних, система переходить до їх обробки. Face-api.js дозволяє виділити та проаналізувати ключові точки обличчя, забезпечуючи високу точність та стабільність даних. Цей процес включає в себе

стабілізацію зображення для мінімізації впливу зовнішніх факторів, таких як зміна кута нахилу обличчя або освітлення. Алгоритми стабілізації дозволяють вирівнювати зображення, що забезпечує більшу точність подальших обчислень.

Однією з ключових функцій системи є генерація пари RSA ключів на основі зчитаних біометричних даних. Цей процес починається з обчислення біометричного відбитка на основі виділених характеристик обличчя. Із цього відбитка формується число, яке стане основою для формування пари асиметричних ключів для шифрування.

Система генерує два числа, які використовуються для формування пари RSA ключів: публічного та приватного. Отримані числа p і q використовуються для обчислення модуля n, який є добутком p і q. Цей модуль є основою для обчислення функції Ейлера φ(n), яка визначається як добуток p-1 і q-1.

Також система генерує симетричний ключ, який зберігається в базі даних H2 та також прийме участь в формуванні пари асинхронних ключів.

Наступним етапом є генерація публічного ключа. Метод генерації публічного ключа використовує значення чисел p і q і байтів симетричного ключа для створення кандидата на публічний ключ. Кандидат обчислюється як сума добутку числа і другого числа, до якого додається симетричний ключ. Потім проводиться перевірка, чи є кандидат взаємно простим з φ(n). Якщо ні, кандидат збільшується на одиницю і перевірка повторюється до тих пір, поки не знайдеться взаємно просте число. Цей кандидат стає публічним ключем.

Останнім кроком є обчислення приватного ключа, який визначається як модульний обернений до публічного ключа по модулю φ(n). Іншими словами, приватний ключ обчислюється так, щоб при множенні на публічний ключ і обчисленні залишку по модулю φ(n) результат був одиницею.

Шифрування та дешифрування даних є центральними функціями системи. Після генерації ключів, дані можуть бути зашифровані за допомогою публічного ключа, що гарантує їхню конфіденційність.

Дешифрування відбувається за допомогою приватного ключа, який також генерується динамічно на основі зчитаних біометричних даних. Такий підхід

забезпечує високий рівень захисту інформації, оскільки приватний ключ існує лише під час сеансу дешифрування.

Система також включає механізми стабілізації зображення та фільтрації шумів для підвищення точності зчитування біометричних даних. Замість одноразового зчитування даних, система збирає метрики протягом певного періоду, наприклад, 15-30 секунд. Це дозволяє вибрати найстабільніші значення для формування біометричного відбитка, що значно підвищує надійність і точність системи. Фільтрація шумів включає використання алгоритмів, які видаляють випадкові артефакти на зображенні, що забезпечує більш точні результати.

Впровадження цих функцій забезпечує надійну роботу системи шифрування на основі біометричних даних обличчя, підвищуючи рівень безпеки та захисту інформації користувачів.

Крім основного функціоналу, система також пропонує користувачам можливість зміни локалі та кольорової схеми інтерфейсу. Користувач може вибрати потрібну мову інтерфейсу з доступних варіантів (українську або англійську), що зручно для не-україномовних користувачів.

Крім того, користувач може змінювати кольорову схему інтерфейсу, вибираючи серед кількох доступних варіантів, що дозволяє налаштувати зовнішній вигляд додатку відповідно до своїх вподобань.

Зміна локалі здійснюється шляхом завантаження відповідних файлів перекладу, після чого інтерфейс додатку автоматично оновлюється згідно з вибраною мовою. Зміна кольорової схеми відбувається за допомогою завантаження відповідного CSS-файлу, що змінює стилі додатку.

## 2.4 Загальні обмеження

Програмна система має певні обмеження, які слід враховувати:

а) обмеження ресурсів:

1) функціональність системи може залежати від ресурсів обчислювального обладнання, на якому вона використовується;

2) для забезпечення оптимальної роботи системи рекомендується використовувати сучасні комп'ютери з достатньою потужністю процесора та пам'яті;

б) енергозабезпечення:

1) важливо забезпечити безперебійне живлення для гарантії постійної роботи системи;

в) зв'язок:

1) для роботи системи необхідне стабільне підключення до мережі Інтернет;

2) перериви в зв'язку можуть призвести до тимчасової недоступності системи;

г) обмеження обсягу шифрування:

1) розроблена програмна система а тому вигляді як вона є генерує ключі довжиною 64 байти, тож обсяг даних для шифрування не має перевищувати 52 байти.

## 2.5 Припущення та залежності

При розробці "Програмної системи" робляться такі припущення:

– користувачі мають доступ до сучасних пристроїв для зчитування біометричних даних, таких як сканери обличчя;
– доступ до мережі Інтернет необхідний для роботи серверних компонентів системи та генерування ключів;
– користувачі повинні мати надійний та швидкісний доступ до Інтернету для забезпечення безперебійної роботи системи;
– для використання Програмної системи користувачі надали свою згоду на зчитування та обробку їхніх біометричних даних.

# 3 АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 UML-проектування програмної системи

Серверна частина проекту реалізовано мовою програмування Java, логіка клієнтської частини додатку - мовою JavaScript.

### 3.1.1 Діаграма розгортання

Наведемо діаграму розгортання, як компоненти нашої системи взаємодіють один з одним та які типи з’єднань використовуються для забезпечення безпеки та ефективності роботи системи (див. рисунок 3.1).

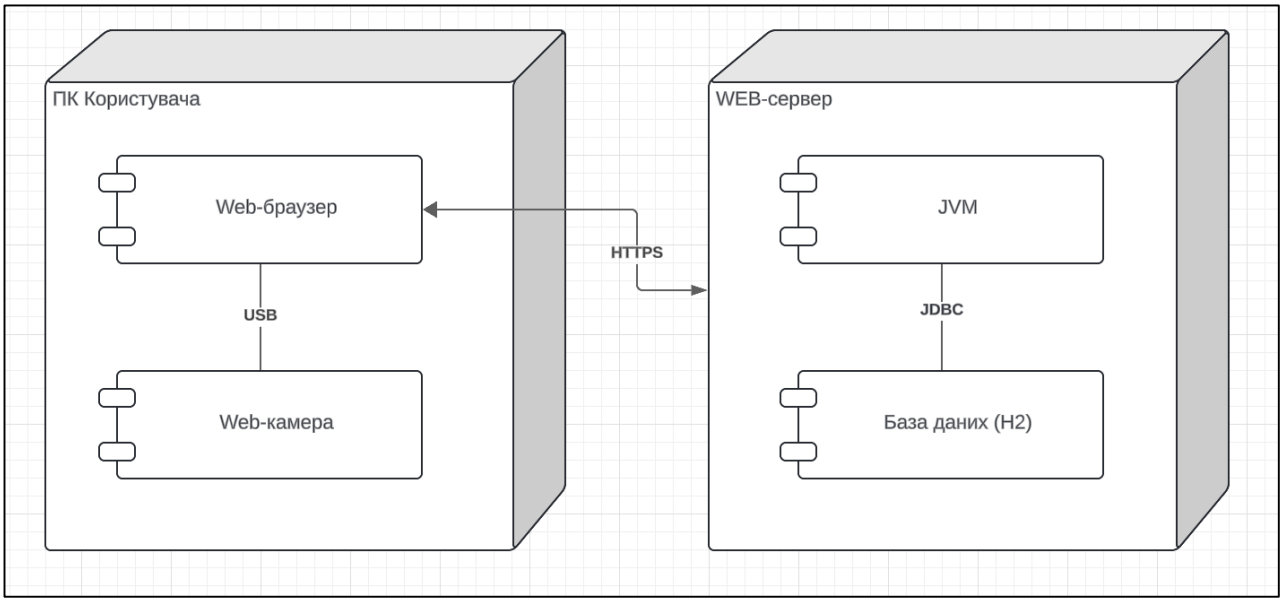



Рисунок 3.1 – Діаграма розгортання програмної системи (рисунок виконаний самостійно)

Діаграма розгортання демонструє фізичне розміщення компонентів програмної системи шифрування на основі біометричних даних обличчя, відображаючи основні вузли системи, їхні компоненти та типи з’єднань між ними. Вона показує, як різні частини системи взаємодіють, забезпечуючи надійну та безпечну роботу.

Клієнтська частина, що запускається у браузері користувача, відповідає за зчитування біометричних даних обличчя за допомогою веб-камери. Використання JavaScript та бібліотеки face-api.js дозволяє зчитувати та попередньо обробляти

біометричні дані. Веб-камера підключена до клієнтської машини, забезпечуючи стабільне та точне зчитування зображень.

Отримані дані передаються на сервер за допомогою протоколу HTTPS, що забезпечує безпечне з'єднання між клієнтом та сервером. Сервер, реалізований на Spring Boot, потребує для розгортання встановлення JVM (Java Virtual Machine). Логіка серверної частину додатку обробляє біометричні дані, виділяє ключові характеристики обличчя та генерує пару RSA ключів. В процесі генерації пари RSA ключів система додатково формує симетричний ключ, байти якого враховуються при формуванні публічного ключа. Цей симетричний ключ зберігається в базі даних H2, яка забезпечує швидкий доступ та ефективне управління даними. Для взаємодії між сервером та базою даних використовується протокол JDBC, що дозволяє зберігати та отримувати дані симетричних ключів.

### 3.1.2 Діаграма прецедентів

Для розуміння взаємодії користувачів із системою та основних функціональних вимог розглянемо Use Case діаграму (діаграма прецедентів) на рисунку 3.2.

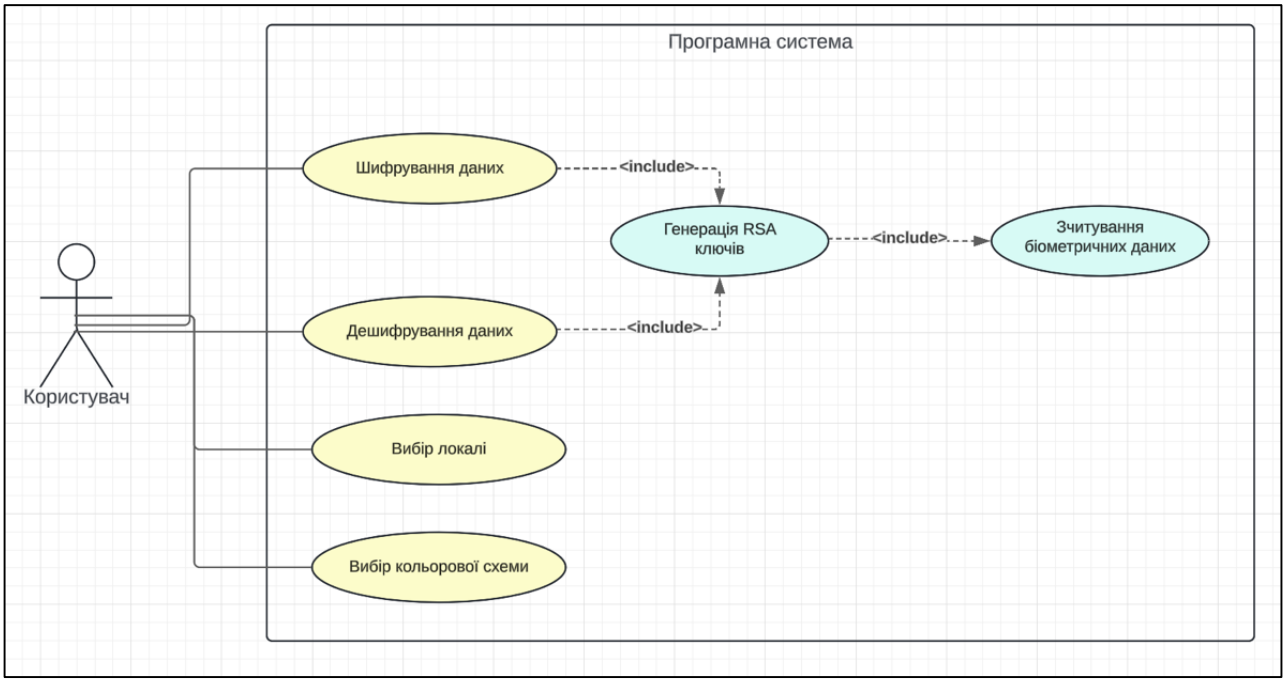



Рисунок 3.2 – Діаграма прецедентів програмної системи (рисунок виконаний самостійно)

Діаграма прецедентів з рисунку 3.2 демонструє, як користувач взаємодіє з системою шифрування на основі біометричних даних обличчя. Вона показує основні функціональні можливості системи та зв’язки між ними, підкреслюючи важливість кожного етапу в загальному процесі шифрування та дешифрування даних.

Розберемо кожен з прецедентів окремо:

- шифрування даних: цей прецедент включає процес шифрування даних за допомогою згенерованого публічного ключа. Користувач може зашифрувати дані, які повинні бути захищені. Включає прецедент “Генерація пари RSA ключів”, оскільки шифрування можливе лише після генерації ключів;
- дешифрування даних: цей прецедент включає процес дешифрування даних за допомогою динамічно згенерованого приватного ключа. Користувач може дешифрувати дані, щоб отримати доступ до їх змісту. Включає прецедент “Генерація пари RSA ключів”, оскільки дешифрування можливе лише після генерації ключів;
- генерація пари RSA ключів: на основі оброблених біометричних даних система генерує пару RSA ключів (публічний та приватний ключі). Включає прецедент “Зчитування біометричних даних”, оскільки генерація ключів можлива лише після отримання біометричних даних;
- зчитування біометричних даних: цей прецедент включає процес зчитування біометричних даних обличчя користувача за допомогою камери;
- вибір локалі: користувач має можливість обрати одну з доступних локалізацій додатку (англійську або українську);
- вибір кольорової схеми: користувач має змогу обрати одну з п’яти запропонованих кольорових схем.

### 3.1.3 Схема алгоритму формування пари RSA ключів

Основою функціонування програм є алгоритм формування пари RSA ключів. Такий алгоритм наведено на рисунку 3.3.


Рисунок 3.3 – Схема алгоритму формування пари RSA ключів в рамках програмної системи (рисунок виконаний самостійно)

Наведена схема демонструє послідовність дій та взаємодію компонентів системи, необхідних для реалізації основних функцій.

Спочатку користувач ініціює процес зчитування біометричних даних обличчя за допомогою веб-камери. Використовуючи JavaScript, клієнтська частина зчитує зображення обличчя та виконує попередню обробку для виділення ключових характеристик. Отримані біометричні дані передаються на сервер.

Оскільки для формування пари RSA ключів потрібно два числа, на сервері на основі отриманих біометричних даних та наявної матриці чисел формується друге число. Проте класичний алгоритм формування RSA ключів вимагає щоб обидва вихідних числа були простими, тож спеціальний метод забезпечує детерміновану генерацію пари простих чисел на основі раніше отриманих.

Для посилення безпеки додатку система формує симетричний ключ, який разом парою вхідних чисел є основою для формування публічного ключа. Дані симетричного ключа зберігаються в базі даних H2 для подальшого використання.

Пара RSA ключів формується на основі детермінованої генерації порстих чисел, обчислення модуля і функції Ейлера, а також підбору публічного ключа і обчислення приватного ключа. Цей процес забезпечує надійність і передбачуваність результатів, необхідних для криптографічних операцій.

### 3.1.4 Діаграма класів

Діаграма класів демонструє структуру основних компонентів і їх взаємодію. На рисунку 3.2 наведено діаграму класів серверної частини розробленої програмної системи.

Для утилітних функцій шифрування та дешифрування даних використовується клас EncryptionUtil. Цей клас відповідає за створення пари RSA ключів, генерацію симетричних ключів, шифрування та дешифрування даних, а також за додавання та витягування метаданих з зашифрованих даних. В ньому також міститься вкладений клас Metadata, який зберігає метадані зашифрованих даних.

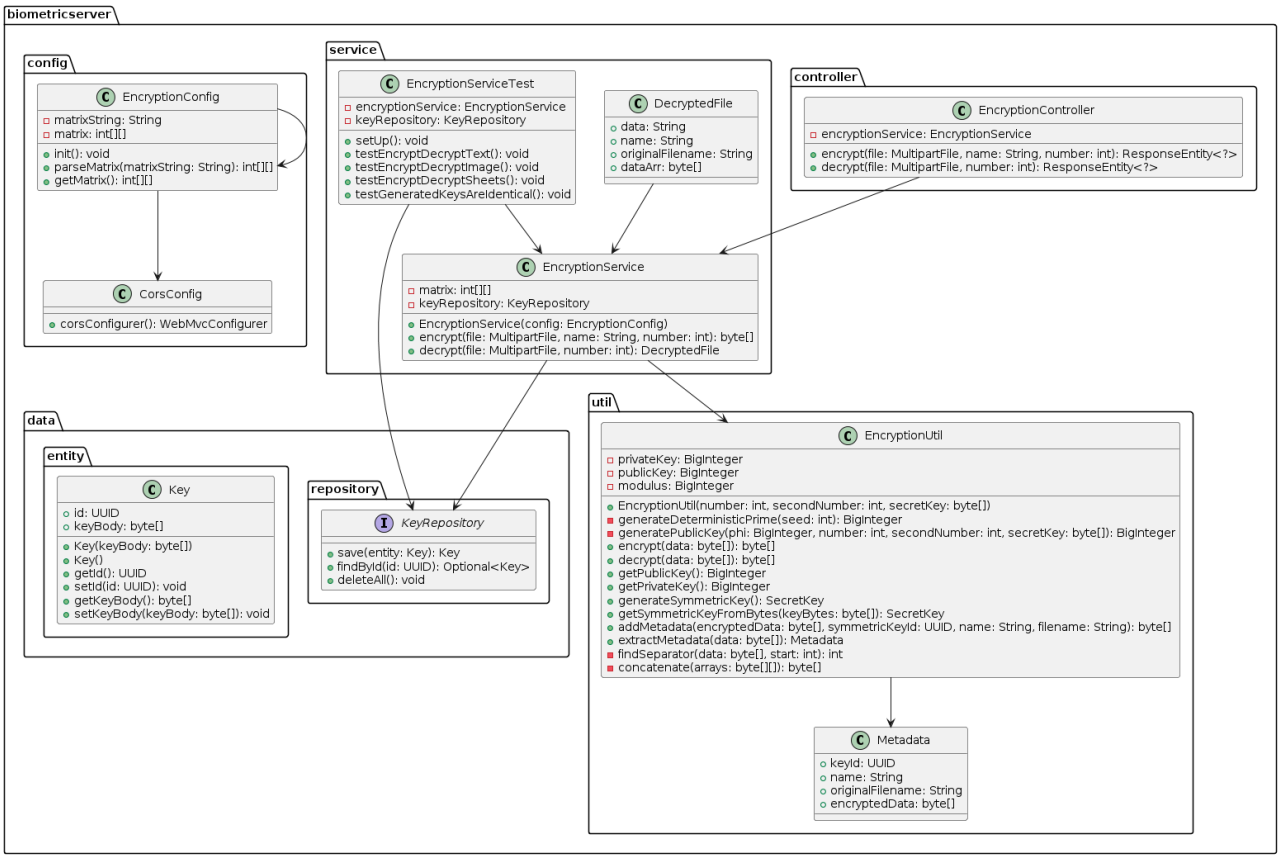



Рисунок 3.4 – Діаграма класів серверної частини програмної системи (рисунок виконаний самостійно)

Сервіс EncryptionService використовує утиліти шифрування для обробки файлів. Він надає методи для шифрування та дешифрування файлів, працюючи з симетричними ключами, що зберігаються в базі даних. Клас EncryptionService також містить внутрішній клас DecryptedFile, який представляє розшифрований файл з його даними, ім’ям та оригінальною назвою.

Для збереження симетричних ключів використовується клас Key, який є JPA сутністю. Цей клас має унікальний ідентифікатор id і байтовий масив, що представляє ключ. Взаємодія з базою даних для сутності Key здійснюється через інтерфейс KeyRepository, який розширює JpaRepository.

Конфігурація шифрування реалізована в класі EncryptionConfig, який зчитує матрицю чисел з файлу налаштувань і забезпечує її доступність через метод getMatrix. Для налаштування CORS політики використовується клас CorsConfig, який забезпечує дозволені налаштування для крос-доменних запитів.

Контролер EncryptionController надає REST API для шифрування та дешифрування файлів. Він має два основних методи: encrypt для шифрування файлу та decrypt для його дешифрування. Обидва методи взаємодіють з сервісом EncryptionService.

Всі ці компоненти разом забезпечують функціональність серверної частини програмної системи, реалізуючи шифрування і дешифрування файлів із використанням асиметричних і симетричних ключів та забезпечуючи збереження ключів у базі даних.

### 3.1.5 Діаграма послідовностей

На рисунку 3.5 розглянемо діаграму послідовностей, яка демонструє процес взаємодії між користувачем, веб-браузером, клієнтською частиною додатку і сервером під час шифрування файлу.

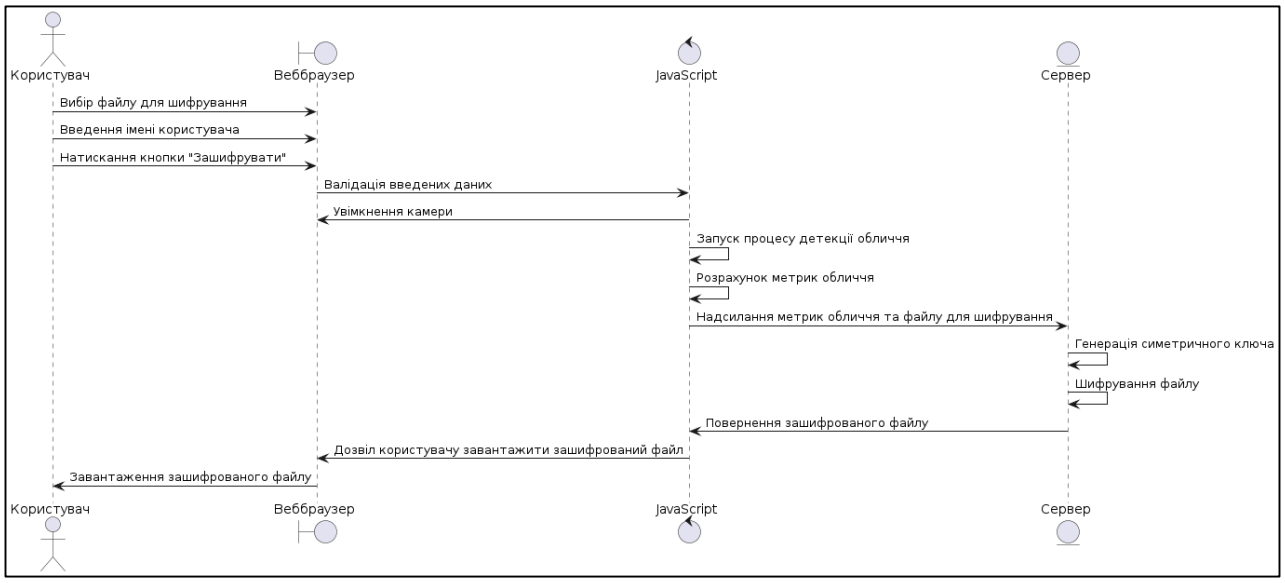



Рисунок 3.5 – Діаграма послідовностей для процесу шифрування файлу (рисунок виконаний самостійно)

Процес починається з того, що користувач вибирає файл для шифрування і вводить своє ім’я. Після цього він натискає кнопку “Зашифрувати”. Веб-браузер приймає ці дії і передає їх клієнтській частині додатку на JavaScript. JavaScript, у свою чергу, перевіряє, чи всі необхідні поля заповнені.

Після успішної перевірки даних, JavaScript вмикає камеру на пристрої користувача і запускає процес детекції обличчя. Відеопотік з камери аналізується для виявлення обличчя користувача. Коли обличчя виявлено, JavaScript розраховує метрики обличчя, такі як пропорції і особливості обличчя, і зберігає їх для подальшого використання.

Далі ці метрики разом з файлом передаються на сервер. Сервер приймає ці дані і генерує пару RSA ключів для шифрування файлу. Потім сервер виконує шифрування файлу і відправляє зашифрований файл назад клієнтському додатку.

Клієнтський додаток приймає зашифрований файл і автоматично завантажує його.

Аналогічно відбувається процес розшифрування файлу. На рисунку 3.6 наведемо діаграму послідовностей цього процесу.

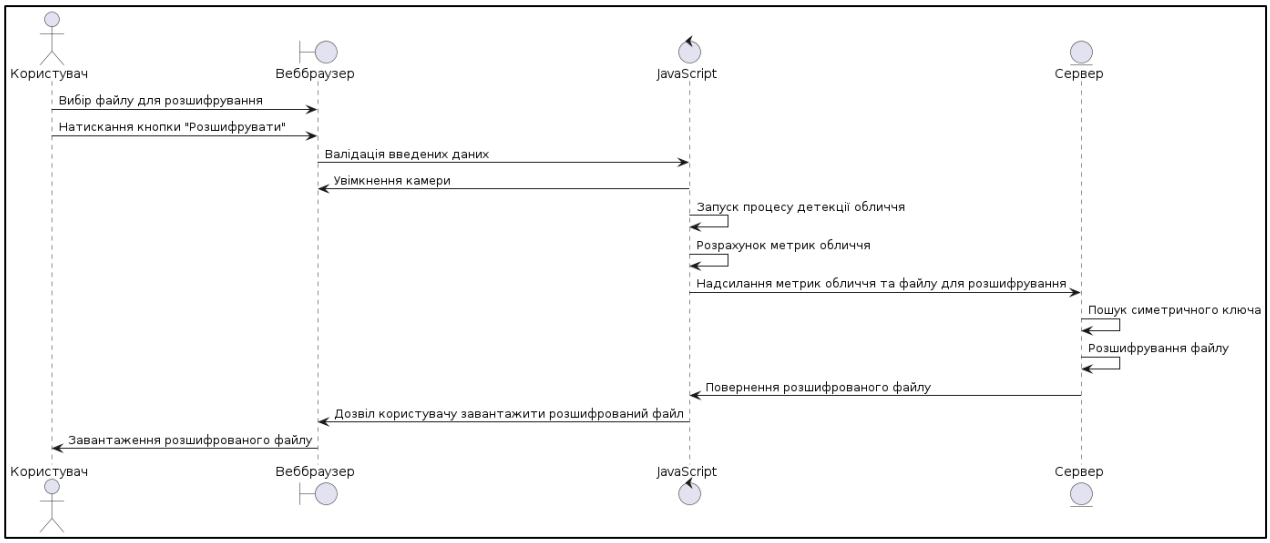



Рисунок 3.6 – Діаграма послідовностей для процесу розшифрування файлу (рисунок виконаний самостійно)

Таким чином, діаграма послідовностей демонструє весь шлях від взаємодії користувача з веб-додатком до завершення процесу шифрування або розшифрування файлу і надання користувачу результату. Цей процес включає кілька етапів перевірки даних, аналізу обличчя і безпечного обміну даними між клієнтом і сервером, що забезпечує високий рівень безпеки і зручності для користувача.

## 3.2 Проектування архітектури програмного забезпечення

Програмна система шифрування на основі біометричних даних обличчя розроблена на основі клієнт-серверної архітектури, яка забезпечує високу продуктивність, безпеку та масштабованість. Серверна частина системи реалізована мовою програмування Java з використанням фреймворку Spring Boot, що дозволяє ефективно обробляти HTTP запити та взаємодіяти з базою даних H2.

Клієнтська частина реалізована мовою програмування JavaScript з використанням бібліотеки face-api.js для зчитування та обробки біометричних даних.

Серверна частина складається з Web API, до якого можливо звернутися через відправку HTTP запитів з тілом у форматі JSON. Відповіді від сервера також надходять у форматі JSON, що забезпечує зручність обробки даних на клієнтській стороні. Web API реалізовано на основі Spring Boot, що забезпечує високу продуктивність та надійність роботи сервера.

Основні компоненти серверної частини:

– контролери: відповідають за обробку вхідних HTTP запитів, виклик відповідних сервісів та формування HTTP відповідей. Контролери отримують дані від клієнта, передають їх на обробку сервісам та повертають результати клієнту;
– сервіси: реалізують бізнес-логіку системи, обробляють дані та виконують основні операції. Сервіси взаємодіють з репозиторіями для збереження та отримання даних з бази даних;
– репозиторії: відповідають за взаємодію з базою даних H2. Вони виконують операції збереження, оновлення, видалення та пошуку даних у базі даних. Репозиторії використовують Hibernate для роботи з базою даних, що забезпечує зручність та ефективність роботи з даними.

## 3.3 Проектування бази даних

Програмний продукт не потребує складної та розгалуженої бази даних, адже генерує RSA-ключі на льоту під час зчитування біометричних даних користувача, а не зберігає їх.

База даних складається з однієї таблиці, яка зберігає тіло згенерованого симетричного ключа, який бере участь в динамічному формуванні пари RSA ключів.

Програмна система не зберігає жодних особистих даних користувачів, таких як біометричні дані обличчя або особиста інформація. Це гарантує, що дані не піддаються ризику витоку або злому, а також мінімізує ризик несанкціонованого доступу до особистої інформації користувачів

# 4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

## 4.1Опис прийнятих інфраструктурних рішень

Під час розробки програмної системи вибір інфраструктурних рішень відштовхувався від питань забезпечення продуктивності, масштабованості, безпеки та зручність використання.

Для розробки серверної частини проекту було обрано використання фреймворку Spring.

Spring – це потужний фреймворк для розробки Java-додатків, який забезпечує інфраструктуру для створення стабільних, масштабованих і гнучких додатків. Він пропонує багатий набір функціональних можливостей для управління залежностями, обробки даних, управління транзакціями, безпеки тощо.

Spring Boot – це розширення Spring, яке спрощує створення автономних, готових до роботи Spring-додатків. Основна мета Spring Boot – зменшити кількість необхідної конфігурації та забезпечити швидкий старт проекту. Spring Boot автоматично налаштовує додаток на основі доданих залежностей.

Одним із ключових аспектів безпеки системи є використання HTTPS протоколу для шифрування трафіку між клієнтом і сервером. HTTPS, що розшифровується як HyperText Transfer Protocol Secure, є розширенням HTTP, яке використовує SSL/TLS протоколи для шифрування даних. Це забезпечує конфіденційність та цілісність переданої інформації.

Важливим елементом фреймворків Spring та Spring Boot є анотації, які значно спрощують конфігурацію та управління додатком. Вони дозволяють розробникам легко додавати метадані до класів, методів та полів, що забезпечує автоматичне налаштування та інтеграцію різних компонентів системи.

Одним із ключових аспектів безпеки системи є використання HTTPS протоколу для шифрування трафіку між клієнтом і сервером. HTTPS, що розшифровується як HyperText Transfer Protocol Secure, є розширенням HTTP, яке використовує SSL/TLS протоколи для шифрування даних.

Для налаштування HTTPS у Spring Boot використовується SSL сертифікат. На рисунку 4.1 наведено команду формування SSL сертифікату.

```
keytool -genkeypair -alias springboot -keyalg RSA -keysize 4096 -storetype PKCS12 -keystore springboot.p12 -validity 3650 -storepass password
```

Рисунок 4.1 – Скрипт формування SSL сертифікату (рисунок виконаний самостійно)

Конфігурація використання SSL здійснюється через файл application.properties (див. рисунок 4.2). Як ми бачимо з рисунку, сервер налаштований на використання порту 8898 для HTTPS з PKCS12 ключовим сховищем, яке містить сертифікат та приватний ключ. Це дозволяє гарантувати, що всі з'єднання між клієнтами та сервером будуть зашифровані, що запобігає перехопленню або зміні даних під час їх передачі.

```
server.port = 8898
server.ssl.key-store = springboot.p12
server.ssl.key-store-password = password
server.ssl.keyStoreType = PKCS12
server.ssl.keyAlias = springboot
```

Рисунок 4.2 – Конфігурація SSL з файлу application.properties (рисунок виконаний самостійно)

Для зберігання симетричних ключів та інших даних використовується вбудована база даних H2. Взаємодія з базою даних здійснюється за допомогою JPA (Java Persistence API) з використанням Spring Data JPA.

JPA – це специфікація для управління реляційними даними в Java. Вона забезпечує засоби для відображення об'єктів Java на записи в базі даних і навпаки. JPA дозволяє розробникам працювати з базами даних на рівні об'єктів, не турбуючись про SQL запити.

Spring Data JPA – це частина екосистеми Spring, яка спрощує реалізацію репозиторіїв для роботи з базами даних. Вона надає інтерфейси та готові реалізації для виконання CRUD операцій, пошуку та інших завдань.

```
spring.datasource.url=jdbc:h2:file:./db/diploma
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.generate-ddl=true
spring.jpa.show-sql=true
```

Рисунок 4.3 – Налаштування бази даних у application.properties (рисунок виконаний самостійно)

Конфігурація бази даних, наведена на малюнку 4.3 визначає URL для підключення до бази даних, драйвер, користувача та пароль, а також налаштування JPA для автоматичного оновлення схеми бази даних.

Необхідно також зазначити, що матриця простих чисел, яка використовується для формування пари RSA ключів також міститься в application.properties (див. рисунок 4.4). Такий підхід дозволяє не використовувати статичну матрицю, а зробити її налаштовуваною.

```
encryption.matrix=11, 13, 17, 19, 23, 29, 31, 37, 41, 43;47, 53, 59, 61, 67, 71, 73, 79, 83, 89;
```

Рисунок 4.4 – Налаштування матриці чисел в application.properties (рисунок виконаний самостійно)

Прийняті інфраструктурні рішення забезпечують високий рівень безпеки, продуктивності та зручності використання системи. Використання HTTPS протоколу гарантує захист даних під час передачі між клієнтом і сервером. Застосування Spring Boot спрощує розробку та конфігурацію серверної частини, а

інтеграція з JPA та Spring Data JPA дозволяє ефективно керувати даними в базі даних. Ці рішення забезпечують надійну та масштабовану інфраструктуру, яка відповідає сучасним вимогам користувачів.

## 4.2 Опис прийнятих рішень для клієнтської частини

При розробці клієнтської частини програмної системи було прийнято кілька важливих інфраструктурних рішень, спрямованих на забезпечення зручності, продуктивності та користувацького досвіду. Нижче наведено детальний опис прийнятих рішень з прикладами коду та поясненням їх впровадження.

Основним файлом клієнтської частини є index.html, який визначає структуру сторінки та включає посилання на необхідні стилі та скрипти. Він містить розділи для вибору режиму (шифрування або розшифрування), завантаження файлу, відображення відео з камери для розпізнавання обличь та завантаження результатів.

Протягом розробки клієнтської частини програмної системи було самостійно розроблено логотип та фавікон за допомогою платформи Figma, що додає додатку індивідуальності та професійного вигляду (див. рис.4.5, 4.6).


Рисунок 4.5 – Логотип програмної системи (рисунок виконаний самостійно)


Рисунок 4.6 – Фавікон порграмної системи (рисунок виконаний самостійно)

Для підтримки багатомовності додатку використовуються JSON файли з перекладами, які завантажуються динамічно залежно від вибраної мови. Функції

для завантаження та застосування перекладів знаходяться в файлі localization.js (див.рис.4.7).


Рисунок 4.7 Приклад реалізації локалізації (рисунок виконаний самостійно)

Користувач може змінювати кольорову схему інтерфейсу, що дозволяє персоналізувати зовнішній вигляд додатку. Вибір кольору зберігається в URL параметрах і застосовується під час завантаження сторінки (див.рис.4.8).


Рисунок 4.8 – Приклад реалізації зміни кольорової схеми (рисунок виконаний самостійно)

Для обробки відео та розпізнавання обличь використовується бібліотека face-api.js. Ця бібліотека забезпечує можливості для детекції обличь, розпізнавання осіб та інших операцій з використанням нейронних мереж.

Клієнтська частина програмної системи реалізована з використанням сучасних технологій та підходів, що забезпечує зручність використання та високу продуктивність. Завдяки підтримці багатомовності та можливості налаштування кольорової схеми, користувачі можуть персоналізувати інтерфейс під свої потреби. Інтеграція з face-api.js дозволяє здійснювати розпізнавання обличчя та обробку відео в режимі реального часу, що робить систему інноваційною та зручною у використанні.

## 4.3 Опис функціональності формування пари RSA ключів

Формування пари RSA ключів є важливим етапом у забезпеченні безпеки даних через шифрування. RSA (Rivest-Shamir-Adleman) – це асиметричний криптографічний алгоритм, який використовує два різних ключі: публічний ключ для шифрування даних і приватний ключ для їх розшифрування. В розробленій програмній системі функціональність формування пари RSA ключів реалізована за допомогою класу EncryptionUtil.

Процес формування пари RSA ключів починається зі створення об’єкта класу EncryptionUtil (див. рисунок 4.9). Цей клас приймає три параметри: два цілі числа (number та secondNumber) і масив байтів симетричного ключа (secretKey). Перше число (number) отримується з біометричних даних обличчя людини. Друге число (secondNumber) вираховується на основі першого числа та матриці простих чисел, наявної в проекті. Ці параметри використовуються для генерації детермінованих простих чисел, необхідних для побудови ключів.

При створенні об’єкта EncryptionUtil спочатку генеруються два простих числа p та q за допомогою методу generateDeterministicPrime. Цей метод використовує цілі числа як початкове значення для генерації SHA-256 хешу. Отриманий хеш перетворюється на велике ціле число (BigInteger), з якого знаходиться найближче ймовірне просте число (див.рис.4.10).

```
4 usages  Olena Havrylenko
public EncryptionUtil(int number, int secondNumber, byte[] secretKey) {
    BigInteger p = generateDeterministicPrime(number);
    BigInteger q = generateDeterministicPrime(secondNumber);
    this.modulus = p.multiply(q);
    BigInteger phi = (p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));

    this.publicKey = generatePublicKey(phi, number, secondNumber, secretKey);
    this.privateKey = publicKey.modInverse(phi);
}
```

Рисунок 4.9 – Приклад ініціалізації класу EncryptionUtil (рисунок виконаний самостійно)

```
private BigInteger generateDeterministicPrime(int seed) {
    try {
        MessageDigest sha256 = MessageDigest.getInstance( algorithm: "SHA-256");
        sha256.update(BigInteger.valueOf(seed).toByteArray());
        byte[] digest = sha256.digest();
        return new BigInteger( signum: 1, digest).nextProbablePrime();
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException("SHA-256 algorithm not found", e);
    }
}
```

Рисунок 4.10 – Приклад формування детермінованого простого числа (рисунок виконаний самостійно)

Прості числа p та q є основою для обчислення модуля n. Модуль n обчислюється як добуток двох простих чисел p та q: n = p * q. Значення n використовується як модуль для публічного та приватного ключів. Далі обчислюється функція Ейлера φ(n), яка визначається як φ(n) = (p - 1) * (q - 1). Ця функція використовується для генерації публічного ключа (див.рис.4.11).

Публічний ключ e обирається так, щоб він був взаємно простим з φ(n), тобто gcd(e, φ(n)) = 1. Це забезпечує, що публічний ключ можна використовувати для шифрування даних. Приватний ключ d обчислюється як мультиплікативна обернена величина публічного ключа e за модулем φ(n): d = e^(-1) mod φ(n). Це

забезпечує, що d є правильним приватним ключем для відповідного публічного ключа e.

```
private BigInteger generatePublicKey(BigInteger phi, int number, int secondNumber, byte[] secretKey) {
        BigInteger bigNumber = BigInteger.valueOf(number);
        BigInteger bigSecondNumber = BigInteger.valueOf(secondNumber);
        BigInteger bigSecretKey = new BigInteger( signum: 1, secretKey);

        BigInteger publicKeyCandidate = bigNumber.multiply(bigSecondNumber).add(bigSecretKey);

        while (!phi.gcd(publicKeyCandidate).equals(BigInteger.ONE)) {
            publicKeyCandidate = publicKeyCandidate.add(BigInteger.ONE);
        }

        return publicKeyCandidate;
}
```

Рисунок 4.11 – Метод генерації публічного ключа (рисунок виконаний самостійно)

Таким чином, процес формування пари RSA ключів в системі забезпечує надійне шифрування та розшифрування даних, використовуючи добре відомий та перевірений алгоритм асиметричного шифрування.

# 5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 5.1 Опис застосованих видів тестування

Тестування є невід'ємною частиною розробки програмного забезпечення, яке забезпечує впевненість у правильності та стабільності системи.

У процесі розробки програмної системи було застосовано два основних види тестування: мануальне тестування та інтеграційне тестування. Вибір цих методів був зумовлений специфікою проекту та потребою забезпечити високу якість програмного продукту.

Мануальне тестування дозволяє тестувальникам виконувати дії в системі так, як це робитимуть кінцеві користувачі, ідентифікуючи будь-які проблеми або дефекти, які можуть виникнути в процесі реального використання. Це особливо важливо для програм, орієнтованих на кінцевих користувачів, де інтерфейс та зручність використання відіграють ключову роль. Мануальне тестування забезпечує гнучкість і можливість оперативно реагувати на зміни в інтерфейсі або логіці програми, дозволяючи виявити проблеми, які можуть бути непомітні в автоматизованих тестах. [12]

Інтеграційне тестування фокусується на перевірці взаємодії між різними компонентами системи. Інтеграційні тести дозволяють перевірити коректну роботу шифрування та розшифрування даних, а також взаємодію з базою даних. Інтеграційні тести дозволяють моделювати реальні сценарії використання системи, що забезпечує більш глибоке та повне тестування функціональності. Цей підхід допомагає виявити проблеми на ранніх стадіях розробки, що знижує вартість їх усунення та підвищує якість кінцевого продукту. [13]

## 5.2 Опис підходів для інтеграційного тестування

У проекті було застосовано метод інтеграційного тестування, щоб перевірити взаємодію між різними компонентами системи. Інтеграційне тестування дозволяє перевірити, чи правильно взаємодіють між собою різні модулі програми і чи функціонують вони відповідно до очікувань. Крім того, обмеження розробленої системи не дозволяють повноцінно протестувати

шифрування всіх типів даних та файлів з різними розширеннями через їхній обсяг. Проте, саме застосування методів інтеграційного тестування дозволяє провести більш повний обсяг тестування функціональності з точки зору шифрування різних типів вхідних файлів.

Представлений на рисунку 5.1 тест перевіряє процес шифрування текстового файлу та його подальше розшифрування. Важливо переконатися, що після шифрування та розшифрування дані залишаються незмінними, а також перевірити правильність роботи з базою даних.

```
@Test
public void testEncryptDecryptText() throws Exception {
    String originalContent = "This is a test file with 1234 $%^&*() content.";
    String userName = "testUser";
    int number = Integer.MAX_VALUE;

    MultipartFile originalFile = new MockMultipartFile(
            name: "file",
            originalFilename: "testfile.txt",
            contentType: "text/plain",
            originalContent.getBytes(StandardCharsets.UTF_8)
    );

    // Encrypt the file
    byte[] encryptedData = encryptionService.encrypt(originalFile, userName, number);
    MultipartFile encryptedFile = new MockMultipartFile(
            name: "file",
            originalFilename: "testfile.txt.enc",
            contentType: "application/octet-stream",
            encryptedData
    );

    // Decrypt the file
    EncryptionService.DecryptedFile decryptedFile = encryptionService.decrypt(encryptedFile, number);
    String decryptedContent = new String(Base64.getDecoder().decode(decryptedFile.getData()), StandardCharsets.UTF_8);

    assertEquals(originalContent, decryptedContent, message: "File content should be identical");
    assertEquals(userName, decryptedFile.getName(), message: "User name should be identical");
    assertEquals(originalFile.getOriginalFilename(), decryptedFile.getOriginalFilename(), message: "File name should be identical");
}
```

Рисунок 5.1 – Тест шифрування та розшифрування текстових даних (рисунок виконаний самостійно)

У тесті наведеному на рисунку 5.2 перевіряється, чи система правильно обробляє випадок невідповідності біометричних даних, що використовуються для шифрування та розшифрування даних. Цей тест імітує ситуацію, коли користувач намагається розшифрувати файл за допомогою біометричних даних, відмінних від тих, що були використані під час шифрування.

```
no usages  new *
@Test
public void testFailDecryptWithDifferentInitNumbers() throws Exception {
    String originalContent = "This is a test file with 1234 $%^&*() content.";
    String userName = "testUser";
    int number = Integer.MAX_VALUE;

    MultipartFile originalFile = new MockMultipartFile(
            name: "file",
            originalFilename: "testfile.txt",
            contentType: "text/plain",
            originalContent.getBytes(StandardCharsets.UTF_8)
    );

    // Encrypt the file
    byte[] encryptedData = encryptionService.encrypt(originalFile, userName, number);
    MultipartFile encryptedFile = new MockMultipartFile(
            name: "file",
            originalFilename: "testfile.txt.enc",
            contentType: "application/octet-stream",
            encryptedData
    );

    // Decrypt the file
    int otherNumber = 12;
    String message = "";
    try {
        encryptionService.decrypt(encryptedFile, otherNumber);
    } catch (Exception exception) {
        System.out.println(exception.getMessage());
        message = exception.getMessage();
    }

    assertTrue(message.contains("error in decryption"));
}
```

Рисунок 5.2 – Тест шифрування та розшифрування за умови наданні різних біометричних даних (рисунок виконаний самостійно)

Тести, представлені на рисунках 5.3 та 5.4 перевіряють шифрування та розшифрування зображень та файлів таблиць, що дозволяє переконатися, що система правильно працює з різними типами файлів.

```java
@Test
public void testEncryptDecryptImage() throws Exception {
    String userName = "testUser";
    int number = Integer.MAX_VALUE;

    ClassPathResource imgFile = new ClassPathResource("testimage.png");
    byte[] imageContent = Files.readAllBytes(imgFile.getFile().toPath());
    byte[] data = Arrays.copyOfRange(imageContent,  from: 0,  to: 53);

    MultipartFile originalFile = new MockMultipartFile(
            name: "file",
            originalFilename: "test.png",
            contentType: "image/png",
            data
    );

    // Encrypt the file
    byte[] encryptedData = encryptionService.encrypt(originalFile, userName, number);
    MultipartFile encryptedFile = new MockMultipartFile(
            name: "file",
            originalFilename: "test.png.enc",
            contentType: "application/octet-stream",
            encryptedData
    );

    // Decrypt the file
    EncryptionService.DecryptedFile decryptedFile = encryptionService.decrypt(encryptedFile, number);

    byte[] decryptedContent = decryptedFile.getDataArr();
    assertArrayEquals(data, decryptedContent,  message: "Image bytes should be identical");
    assertEquals(userName, decryptedFile.getName(),  message: "User name should be identical");
    assertEquals(originalFile.getOriginalFilename(), decryptedFile.getOriginalFilename(),  message: "File name should be identical");
}
```

Рисунок 5.3 – Тест шифрування та розшифрування даних зображення (рисунок виконаний самостійно)

```java
@Test
public void testEncryptDecryptSheets() throws Exception {
    String userName = "testUser";
    int number = Integer.MAX_VALUE;

    ClassPathResource imgFile = new ClassPathResource("testsheets.xls");
    byte[] imageContent = Files.readAllBytes(imgFile.getFile().toPath());
    byte[] data = Arrays.copyOfRange(imageContent,  from: 0,  to: 53);

    MultipartFile originalFile = new MockMultipartFile(
            name: "file",
            originalFilename: "test.xls",
            contentType: "application/vnd.ms-excel",
            data
    );

    // Encrypt the file
    byte[] encryptedData = encryptionService.encrypt(originalFile, userName, number);
    MultipartFile encryptedFile = new MockMultipartFile(
            name: "file",
            originalFilename: "test.xls.enc",
            contentType: "application/octet-stream",
            encryptedData
    );

    // Decrypt the file
    EncryptionService.DecryptedFile decryptedFile = encryptionService.decrypt(encryptedFile, number);

    byte[] decryptedContent = decryptedFile.getDataArr();
    assertArrayEquals(data, decryptedContent,  message: "File bytes should be identical");
    assertEquals(userName, decryptedFile.getName(),  message: "User name should be identical");
    assertEquals(originalFile.getOriginalFilename(), decryptedFile.getOriginalFilename(),  message: "File name should be identical");
}
```

Рисунок 5.4 – Тест шифрування та розшифрування табличних даних (рисунок виконаний самостійно)

Тест, що наведено на рисунку 5.5 перевіряє, чи однакові вхідні дані призводять до генерації однакових публічних і приватних ключів. Це важливо для забезпечення детермінованості генерації ключів на основі заданих параметрів.

```java
@Test
public void testGeneratedKeysAreIdentical() throws Exception {
    int firstNumber = 42;
    int secondNumber = 131;

    SecretKey symmetricKey = EncryptionUtil.generateSymmetricKey();
    byte[] symmetricKeyBytes = symmetricKey.getEncoded();

    // Save symmetric key to database
    Key keyEntity = new Key(symmetricKeyBytes);
    keyRepository.save(keyEntity);
    UUID symmetricKeyId = keyEntity.getId();

    EncryptionUtil encryptionUtilForEncryption = new EncryptionUtil(firstNumber, secondNumber, keyEntity.getKeyBody());

    EncryptionUtil encryptionUtilForDecryption = null;
    Optional<Key> optionalKey = keyRepository.findById(symmetricKeyId);
    if (optionalKey.isPresent()) {
        byte[] symmetricKeyBytesDec = optionalKey.get().getKeyBody();
        assertArrayEquals(symmetricKeyBytes, symmetricKeyBytesDec);

        encryptionUtilForDecryption = new EncryptionUtil(firstNumber, secondNumber, symmetricKeyBytesDec);
    } else {
        throw new IOException("Symmetric key not found");
    }

    assertEquals(encryptionUtilForEncryption.getPublicKey(), encryptionUtilForDecryption.getPublicKey(), message: "Public keys should be identical");
    assertEquals(encryptionUtilForEncryption.getPrivateKey(), encryptionUtilForDecryption.getPrivateKey(), message: "Private keys should be identical");
}
```

Рисунок 5.5 – Тест генерації однакових ключів (рисунок виконаний самостійно)

На рисунку 5.6 наведено приклад результату запуску вищенаведених інтеграційних тестів.

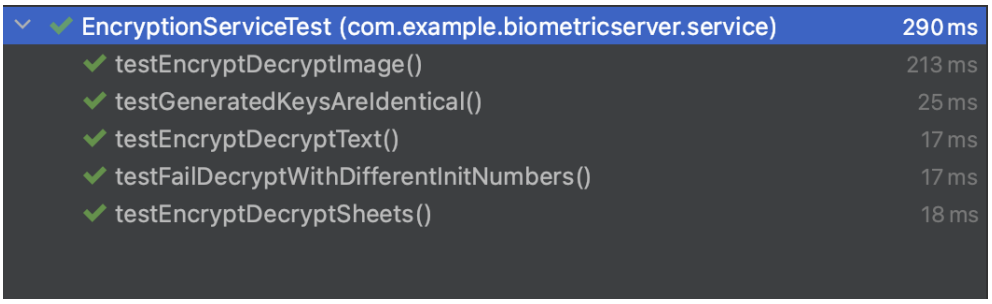



Рисунок 5.6 – Результат запуску інтеграційних тестів (рисунок виконаний самостійно)

Інтеграційне тестування в забезпечує впевненість у правильній роботі всіх компонентів системи та їх взаємодії. Тести перевіряють не тільки правильність шифрування та розшифрування даних, але й взаємодію з базою даних, що забезпечує надійну та стабільну роботу системи в реальних умовах.

## 5.3 Опис підходів для мануального тестування

Мануальне тестування є невід'ємною частиною розробки програмного забезпечення, особливо в проектах, де безпека та надійність відіграють критичну роль. У випадку з біометричною системою шифрування, де конфіденційність даних користувачів залежить від стійкості алгоритмів шифрування та точності зчитування біометричних даних, ретельне ручне тестування стає ще більш важливим.

Мануальне тестування включає в себе перевірку функціональності системи вручну, без використання автоматизованих інструментів. Цей підхід дозволяє тестувальникам виконувати дії в системі так, як це робитимуть кінцеві користувачі, ідентифікуючи будь-які проблеми або дефекти, які можуть виникнути в процесі реального використання. [12]

Для розробленої програмгної системи провдення мануального тестування є особливо важливим, оскільки головною ідеєю проекту є та що шифрування та дешифрування файлів здійснюються за допомогою біометричних даних.

В таблицях 5.1 – 5.4 наведено приклад тест-кейсів, складених під час проведення мануального тестування рорробленої програмної системи.

Тест-кейси, представлені в таблицях 5.1 та 5.2 спрямовані на перевірку базових процесів шифрування та розшифрування файлу. Ітогом проведення обох кейсів має бути розшифрований файл, вміст якого співпадає зі вмістом файлу, переданого до шифрування.

Таблиця 5.1 – Тест-кейс перевірки шифрування файлу

| Id Перевірка шифрування файлу | | | |
|---|---|---|---|
| №пп | Кроки відтворення | Додаткова інформація | Результат проходження кейсу |
| 1 | Попередні налаштування/кроки:<br>-знаходимось на сторінці початку | | pass |

Кінець таблиці 5.1

| Id Перевірка шифрування файлу | | | |
|---|---|---|---|
| №пп | Кроки відтворення | Додаткова інформація | Результат проходження кейсу |
| | шифрування/розшифрування;<br>- у додатку є доступ на роботу з камерою;<br>- у додатку є доступ на роботу з файловою системою користувача;<br>- налаштовано автоматичне завантаження файлів у необхідну теку; | | |
| 2 | Натиснути на кнопку «зашифрувати» | | |
| 3 | Обрати файл для шифрування розміром до 52kB | | |
| 4 | Надати назву файлу, який буде повернуто після шифрування | test.txt | |
| 5 | Натиснути на кнопку «зашифрувати» | | |
| 6 | Після зчитування обличчя через камеру за 15 секунд перейти до теки збереження зашифрованого файлу | | |
| 7 | Зашифрований файл збережено з назвою, заданою при шифруванні та файл має розширення .enc | test.txt.enc | |

Таблиця 5.2 – Тест-кейс перевірки розшифровки зашифрованого файлу

| Id Перевірка розшифровки зашифрованого файлу | | | |
|---|---|---|---|
| №пп | Кроки відтворення | Додаткова інформація | Результат проходження кейсу |
| 1 | Попередні налаштування/кроки:<br>-знаходимось на сторінці початку шифрування/розшифрування;<br>- у додатку є доступ на роботу з камерою;<br>- у додатку є доступ на роботу з файловою системою користувача;<br>- налаштовано автоматичне завантаження файлів у необхідну теку;<br>- успішно виконано кейс «Id Перевірка шифрування файлу» ; | | pass |
| 2 | Натиснути на кнопку «розшифрувати» | | |
| 3 | Обрати файл для розшифровування з теки з назвою test.txt.enc | | |
| 4 | Натиснути на кнопку «розшифрувати» | | |
| 5 | Після зчитування обличчя через камеру за 15 секунд нажати на кнопку «завантажити розшифрований файл» | | |
| 6 | У налаштованій теці буде збережено розшифрований файл з назвою test.txt | | |
| 7 | Виконати перевірку співпадіння вмісту файлів. | Вміст файлу співпадає з файлом до шифрування | |

Таблиця 5.3 – Тест-кейс перевірки завантаження файлу більшого розміру

| Id Перевірка завантаження файлу більшого розміру | | | |
|---|---|---|---|
| №пп | Кроки відтворення | Додаткова інформація | Результат проходження кейсу |
| 1 | Попередні налаштування/кроки:<br>-знаходимось на сторінці початку шифрування/розшифрування;<br>- у додатку є доступ на роботу з камерою;<br>- у додатку є доступ на роботу з файловою системою користувача;<br>- налаштовано автоматичне завантаження файлів у необхідну теку; | | pass |
| 2 | Натиснути на кнопку «зашифрувати» | | |
| 3 | Обрати файл для шифрування розміром більше ніж 52kB | | |
| 4 | Надати назву файлу, який буде повернуто після шифрування | | |
| 5 | Натиснути на кнопку «зашифрувати» | | |
| 6 | Після зчитування обличчя через камеру за 15 секунд сервер поверне помилку | Завантаження файлу не відбулось, файл зашифровано не було | |

Таблиця 5.4 – Тест-кейс перевірки розшифровки зашифрованого файлу іншими біометричними даними

| Id Перевірка розшифровки зашифрованого файлу іншими біометричними даними | | | |
|---|---|---|---|
| №пп | Кроки відтворення | Додаткова інформація | Результат проходження кейсу |
| 1 | Попередні налаштування/кроки:<br>-знаходимось на сторінці початку шифрування/розшифрування;<br>- у додатку є доступ на роботу з камерою;<br>- у додатку є доступ на роботу з файловою системою користувача;<br>- налаштовано автоматичне завантаження файлів у необхідну теку;<br>- успішно виконано кейс «Id Перевірка шифрування файлу» ;<br>- для розшифрування використовуємо біометрію лиця людини, які не брали участі у шифрування; | Обличчя людини, яка не брала участь у шифруванні | pass |
| 2 | Натиснути на кнопку «розшифрувати» | | |
| 3 | Обрати файл для розшифровування з теки з назвою test.txt.enc | | |
| 4 | Натиснути на кнопку «розшифрувати» | | |
| 5 | Після зчитування обличчя через камеру за 15 секунд відобразиться помилка «Decryption failed. Please check uploaded data and try again» | | |

Таблиця 5.4 містить тест-кейс, який перевіряє неможливість розшифрувати зашифрований файл біометричними даними іншої людини. Цей тест вважається пройденим в разі, якщо сервер при спробі розшифрувати файл біометричними даними іншої людини повертає помилку, про те що йому було передано не валідні дані.

Мануальне тестування є важливим етапом розробки програмного забезпечення, особливо в проектах, де надійність та безпека відіграють критичну роль. У випадку з біометричною системою шифрування ретельне ручне тестування гарантує точність зчитування біометричних даних, стійкість алгоритмів шифрування та зручність використання для кінцевих користувачів, тим самим роблячи систему безпечною та доступною.

# ВИСНОВКИ

В результаті виконання роботи було спроектовано та створено програмну систему шифрування інформації на основі біометричних даних обличчя людини.

В ході дослідження було вирішено низку технічних проблем, пов’язаних зі стабільністю та точністю зчитування біометричних даних. Розроблені методи стабілізації зображення та фільтрації шумів, а також алгоритми динамічного формування приватного ключа забезпечують високу точність і надійність роботи системи. Використання сучасних технологій, таких як JavaScript та бібліотеки face-api.js для клієнтської частини, Java, Spring Boot та Hibernate для серверної частини, дозволило створити продуктивну та масштабовану систему.

Структура бази даних H2, що використовується для зберігання симетричних ключів, які беруть участь в процесі формуваня пари RSA ключів, забезпечує простоту та ефективність управління даними, необхідними для шифрування та дешифрування.

У процесі розробки було проведено порівняння та аналіз існуючих аналогів програмних систем шифрування. Було виявлено, що наша система має кілька ключових переваг, серед яких висока ступінь автоматизації процесу генерації ключів, динамічна обробка біометричних даних та використання сучасних технологій для забезпечення стабільності та точності зчитування. Аналіз аналогів показав, що більшість існуючих систем мають обмежені можливості у сфері біометричної аутентифікації та не забезпечують достатньо високого рівня безпеки.

Перспективи подальшого розвитку системи включають вдосконалення алгоритмів обробки біометричних даних, розширення функціональних можливостей та інтеграцію з іншими системами захисту інформації.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Biometric facial recognition – Enhancing user verification and authentication. [Електронний ресурс] URL: https://www.fraud.com/post/biometric-facial-recognition (дата звернення: 20.05.2024).

2. Face Recognition with Local Binary Patterns. [Електронний ресурс] URL: https://www.researchgate.net/publication/221304831_Face_Recognition_with_Local_Binary_Patterns (дата звернення: 15.05.2024).

3. face-api.js – JavaScript API for Face Recognition in the Browser with tensorflow.js. [Електронний ресурс] URL: https://itnext.io/face-api-js-javascript-api-for-face-recognition-in-the-browser-with-tensorflow-js-bcc2a6c4cf07 (дата звернення: 12.05.2024).

4. Realtime JavaScript Face Tracking and Face Recognition using face-api.js' MTCNN Face Detector. [Електронний ресурс] URL: https://itnext.io/realtime-javascript-face-tracking-and-face-recognition-using-face-api-js-mtcnn-face-detector-d924dd8b5740 (дата звернення: 12.05.2024).

5. RSA algorithm (Rivest-Shamir-Adleman). [Електронний ресурс] URL: https://www.techtarget.com/searchsecurity/definition/RSA (дата звернення: 21.05.2024).

6. Security Engineering: A Guide to Building Dependable Distributed Systems. [Електронний ресурс] URL: https://www.cl.cam.ac.uk/~rja14/book.html (дата звернення: 23.05.2024).

7. What are the pros and cons of Face ID on Apple phones? How secure is Face ID? What are its limitations? URL: https://www.quora.com/What-are-the-pros-and-cons-of-Face-ID-on-Apple-phones-How-secure-is-Face-ID-What-are-its-limitations (дата звернення: 17.05.2024).

8. ДСТУ 3008:2015 Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. К.: ДП «УкрНДНЦ», 2016. 26 с. (дата звернення: 01.05.2024).

9. Біометрична ідентифікація: як вона працює. [Електронний ресурс] URL: https://fintechinsider.com.ua/biometrychna-identyfikacziya-yak-vona-praczyuye/ (дата звернення: 07.05.2024).

10. Криптографія з відкритим ключем та цифрові конверти, різновидності алгоритмів, порівняння алгоритмів. [Електронний ресурс] URL: https://sites.google.com/view/blog-ua/%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%96-%D0%BF%D0%BE%D0%BD%D1%8F%D1%82%D1%82%D1%8F-%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D1%96%D1%97-%D1%82%D0%B0-%D0%B7%D0%B0%D1%85%D0%B8%D1%81%D1%82%D1%83-%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D1%97/%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D1%96%D1%8F-%D0%B7-%D0%B2%D1%96%D0%B4%D0%BA%D1%80%D0%B8%D1%82%D0%B8%D0%BC-%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BC-%D1%80%D1%96%D0%B7%D0%BD%D0%BE%D0%B2%D0%B8%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D1%96-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC (дата звернення: 20.05.2024).

11. Розпізнавання обличчя. [Електронний ресурс] URL: https://ukeywaf.com/baza/rozpiznavannya-oblychchya/ (дата звернення: 13.05.2024).

12. Тестування. Фундаментальна теорія. [Електронний ресурс] URL: https://dou.ua/forums/topic/13389/ (дата звернення 10.06.2024).

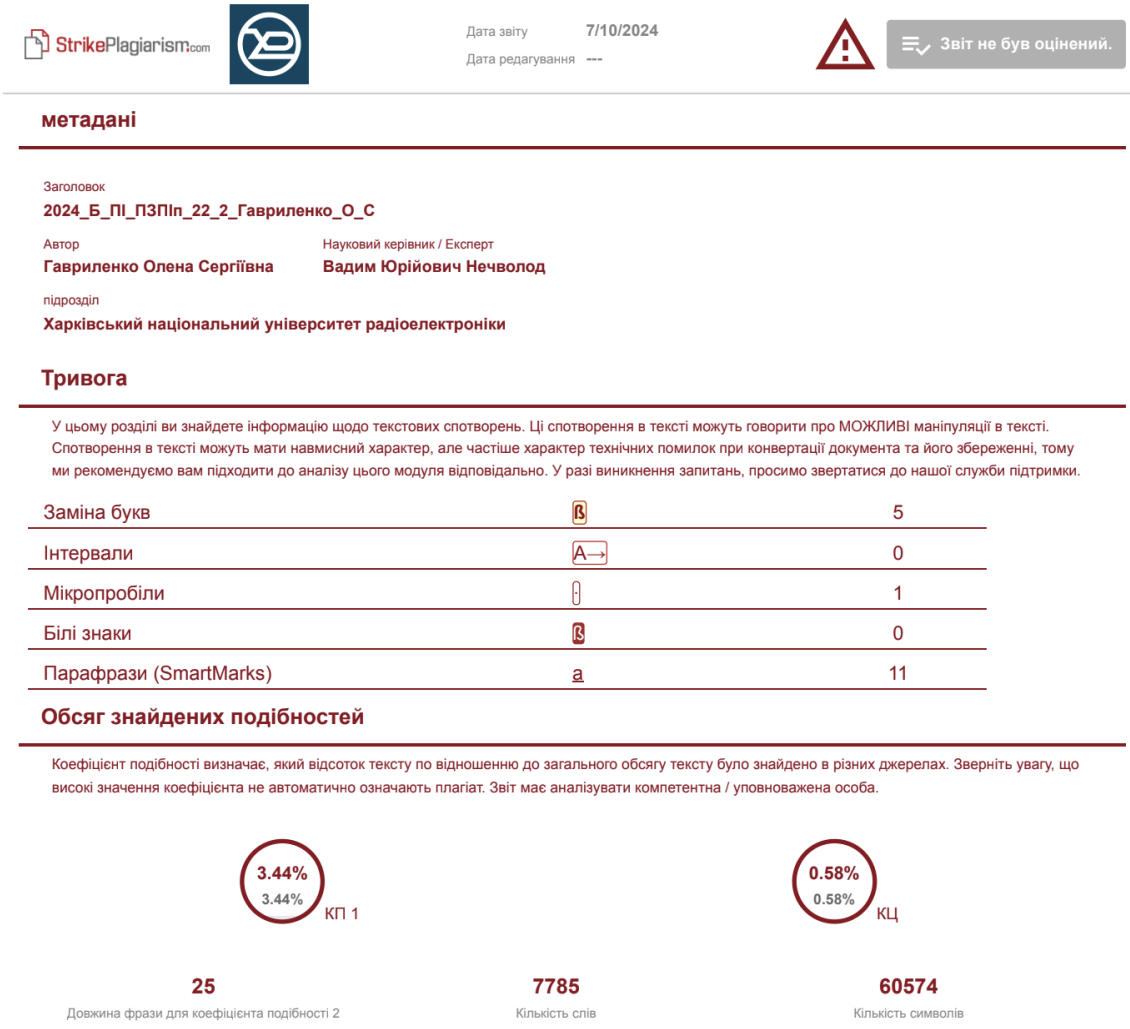
13. Теорія тестування від А до Я. [Електронний ресурс] URL: https://ilarionhalushka.github.io/ua/testing-theory (дата звернення 10.06.2024).

14. Як будувати UML-діаграми. Розбираємо три найпопулярніші варіанти. [Електронний ресурс] URL: https://dou.ua/forums/topic/40575/ (дата звернення: 03.06.2024).

15. Як працює Дія.Підпис і для чого він потрібний. [Електронний ресурс] URL: https://finance.ua/ua/saving/jak-pracue-dia-pidpys (дата звернення: 16.05.2024).

# ДОДАТОК А

## Результат перевірки на плагіат



StrikePlagiarism.com

Дата звіту 7/10/2024
Дата редагування ---

Звіт не був оцінений.

**метадані**

Заголовок
**2024_Б_ПІ_ПЗПІп_22_2_Гавриленко_О_С**

Автор
**Гавриленко Олена Сергіївна**

Науковий керівник / Експерт
**Вадим Юрійович Нечволод**

підрозділ
**Харківський національний університет радіоелектроніки**

**Тривога**

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

| | | |
|---|---|---|
| Заміна букв | ß | 5 |
| Інтервали | A→ | 0 |
| Мікропробіли | · | 1 |
| Білі знаки | ß | 0 |
| Парафрази (SmartMarks) | a | 11 |

**Обсяг знайдених подібностей**

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

3.44% 3.44% КП 1

0.58% 0.58% КЦ

**25**
Довжина фрази для коефіцієнта подібності 2

**7785**
Кількість слів

**60574**
Кількість символів

# ДОДАТОК Б

## Приклади користувацького інтерфейсу

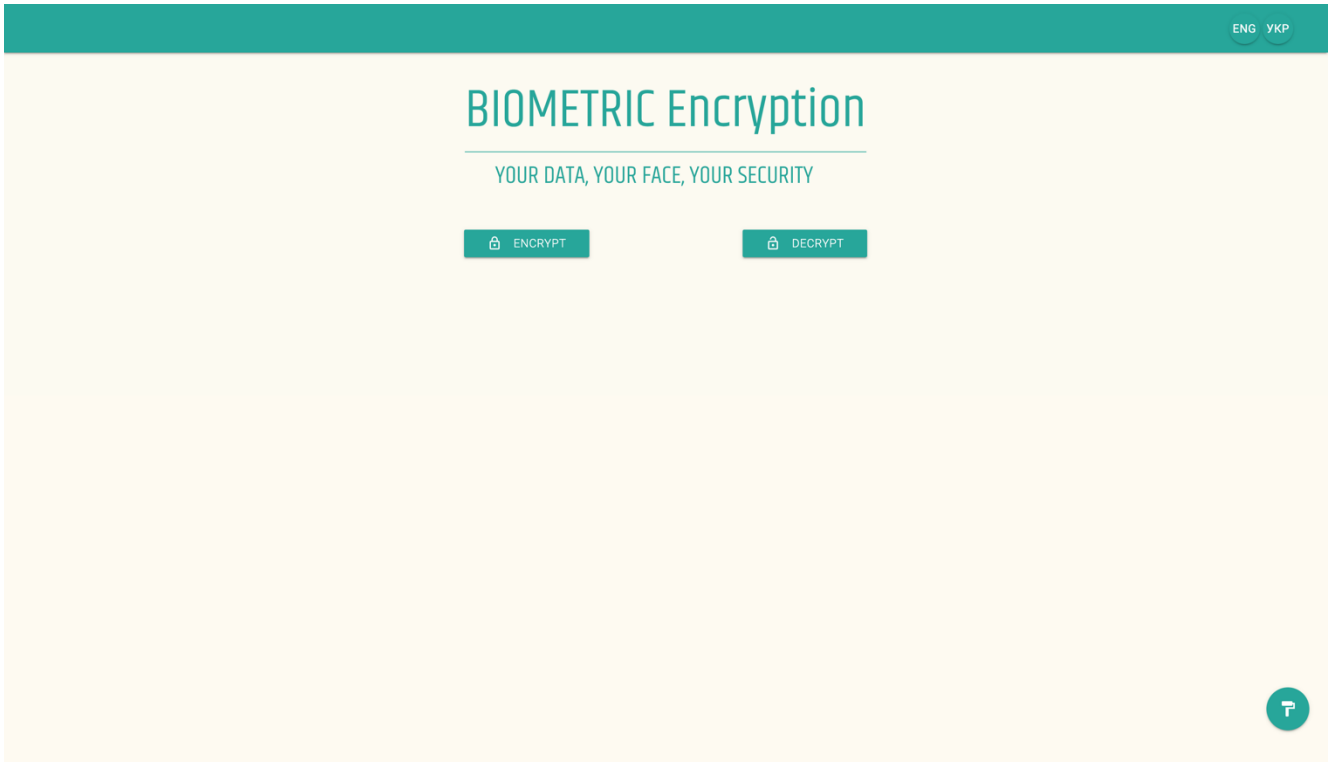



Рисунок Б.1 – Головна сторінка (основна кольорова схема)

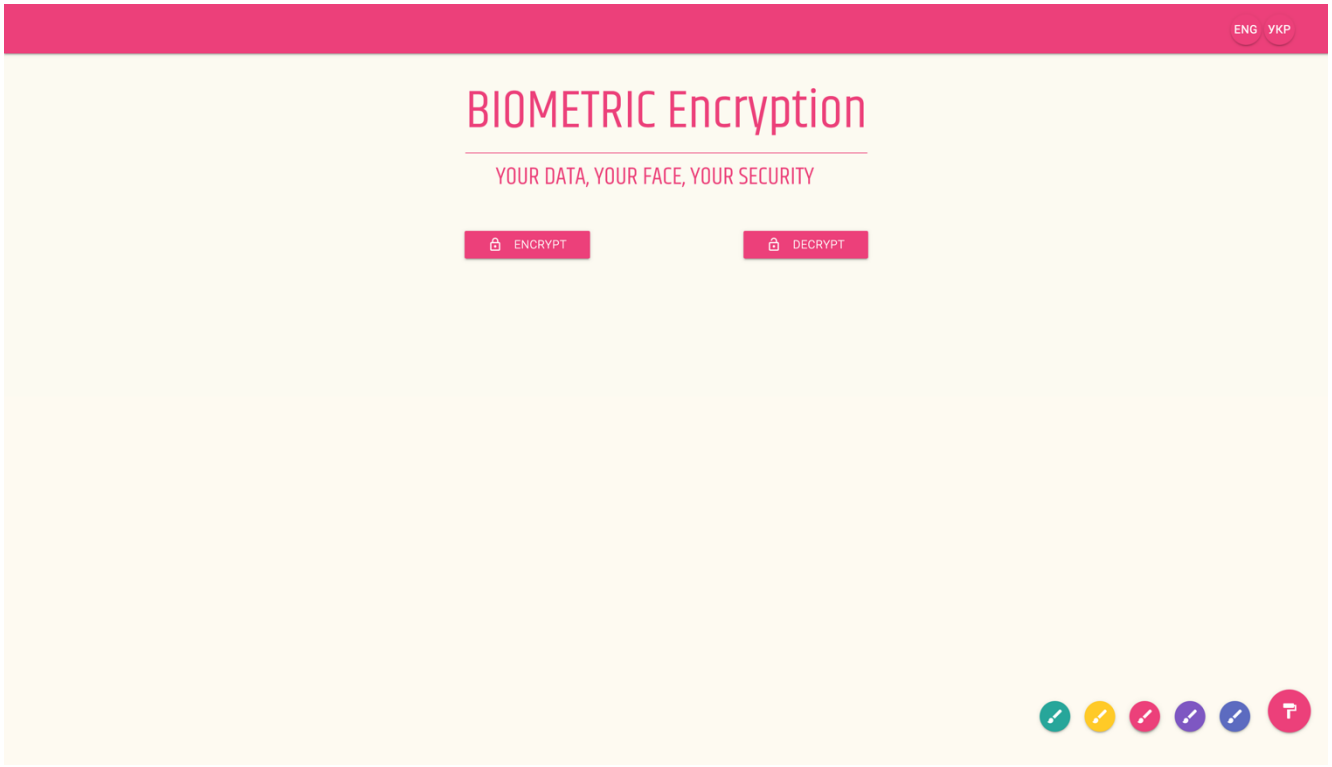



Рисунок Б.2 – Головна сторінка (рожева кольорова схема)

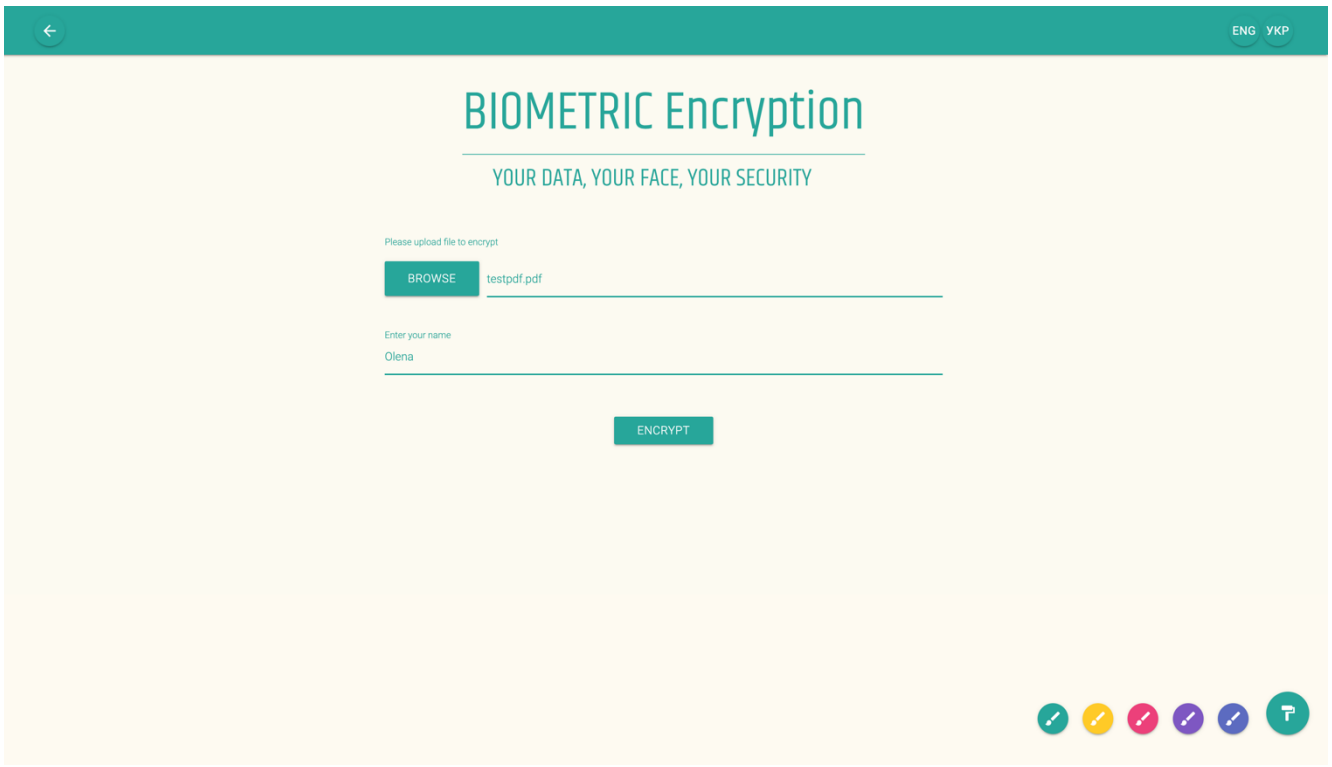



Рисунок Б.3 – Сторінка шифрування файлу

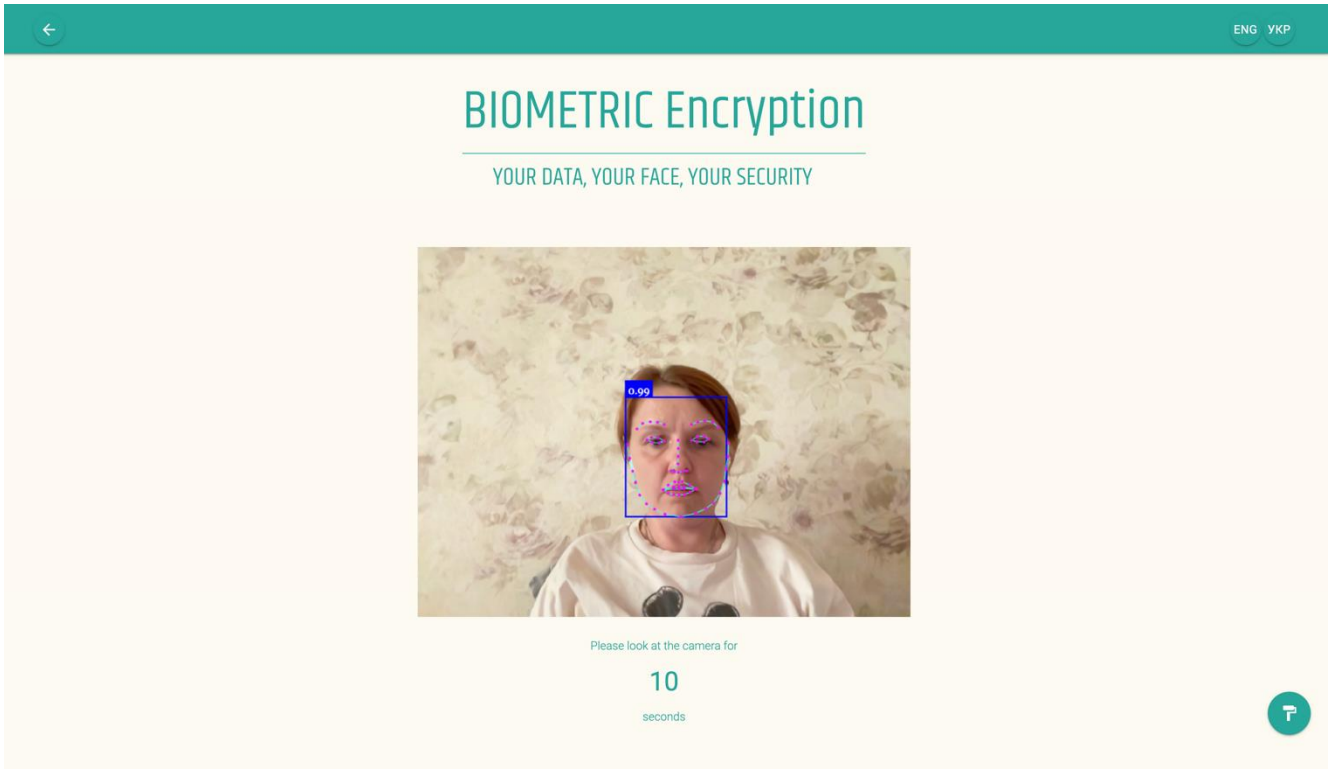



Рисунок Б.4 – Сторінка зчитування біометричних даних обличчя

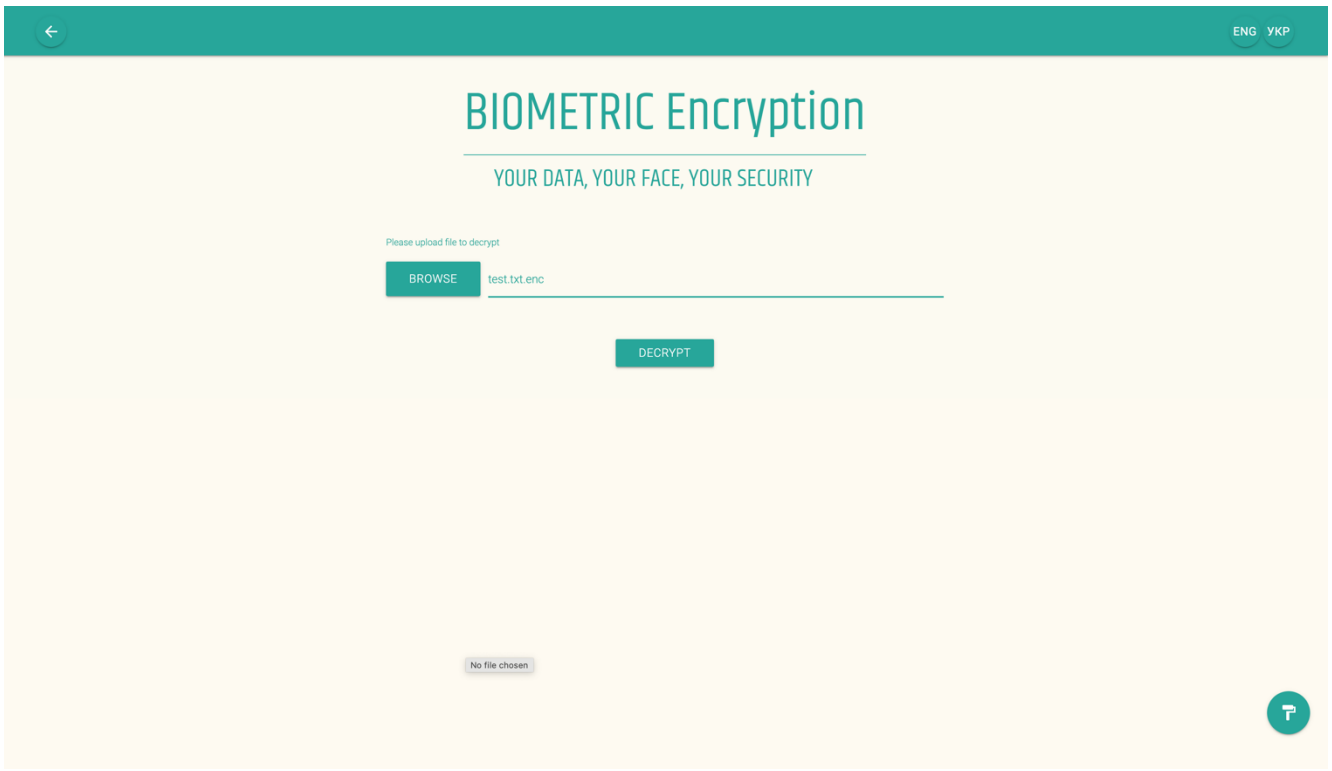



Рисунок Б.5 – Сторінка завантаження файлу на розшифрування

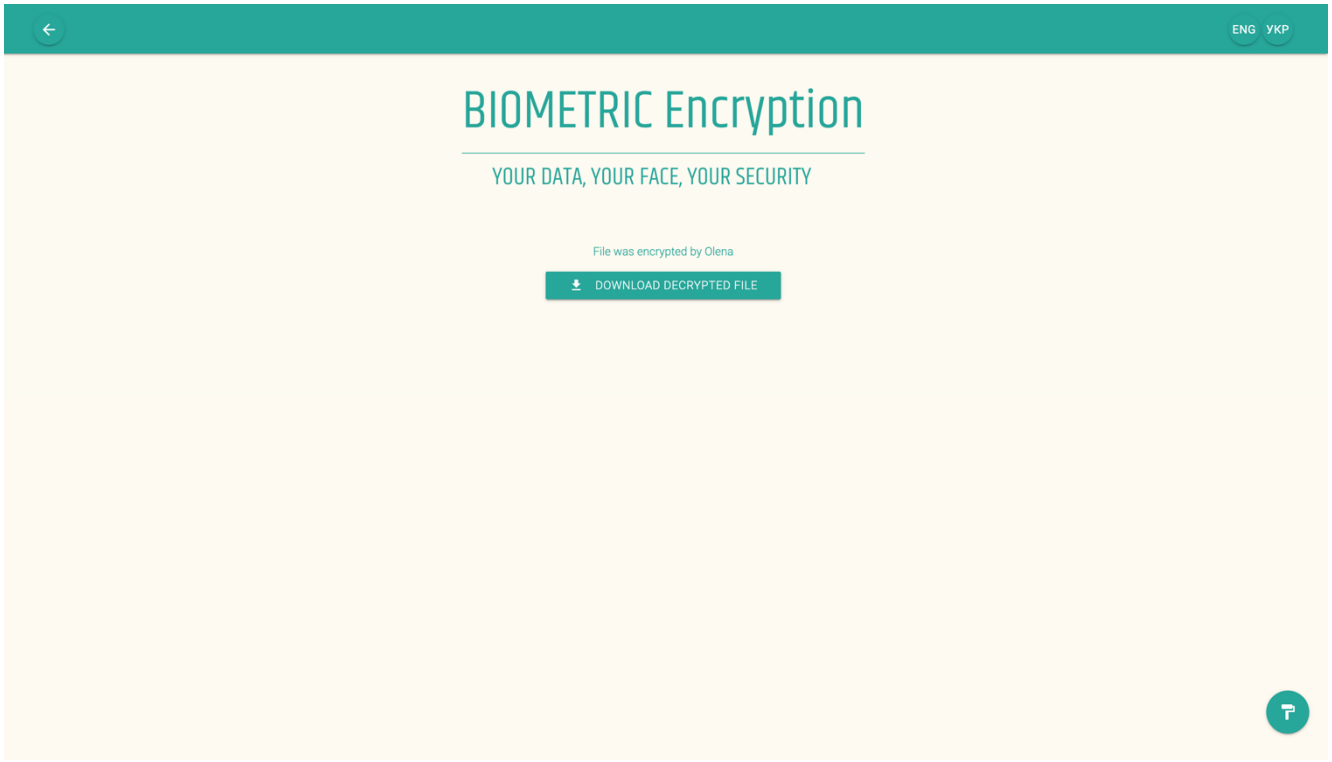



Рисунок Б.6 – Сторінка завантаження розшифрованого файлу

# ДОДАТОК В

## Слайди презентації

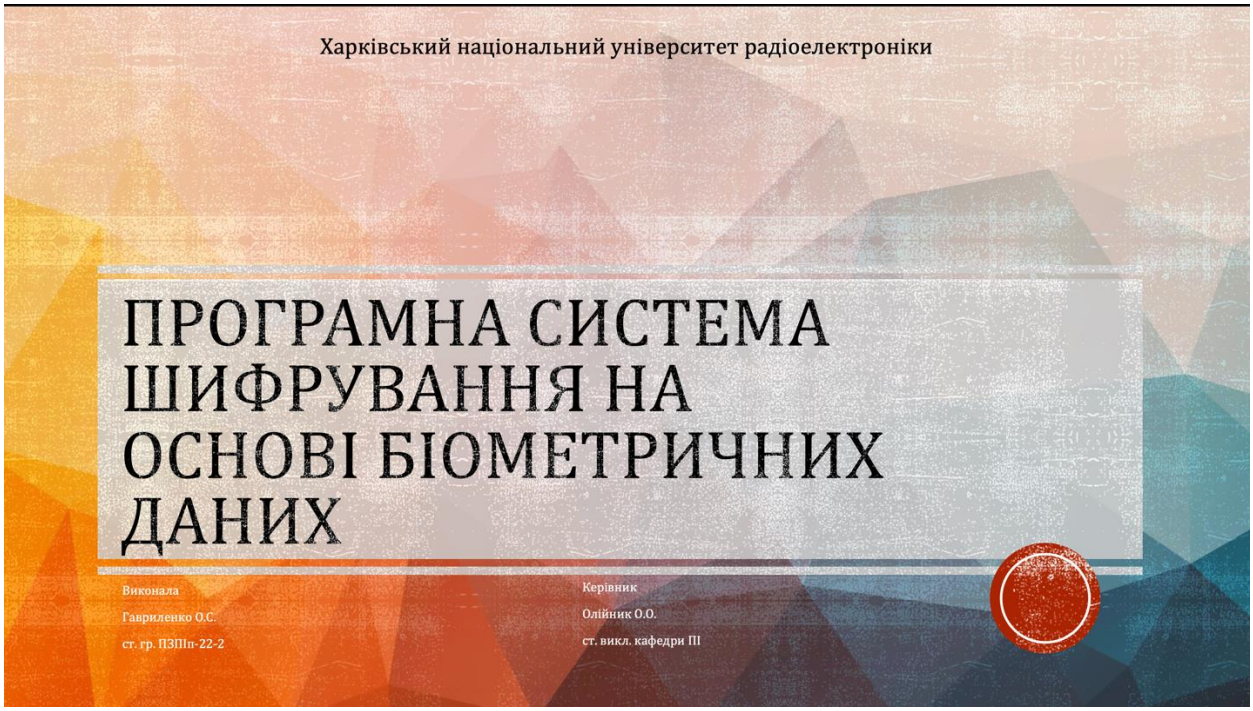



Рисунок В.1 – Слайд 1

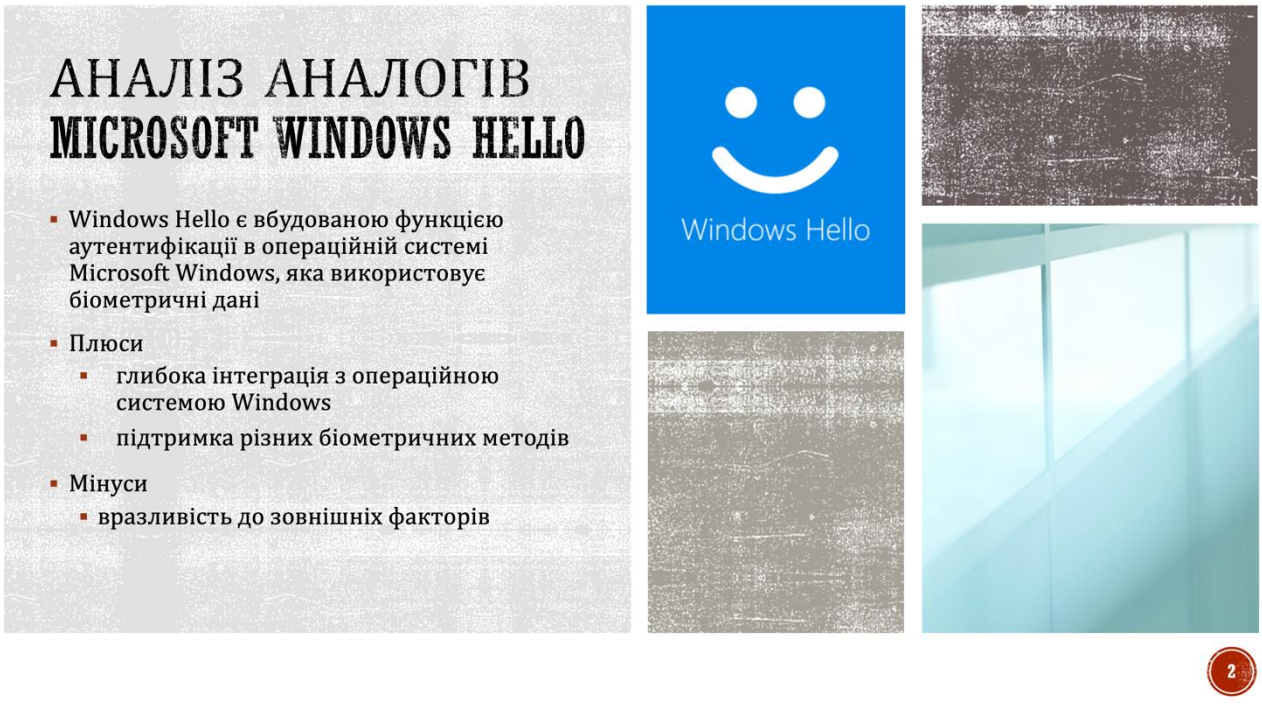



Рисунок В.2 – Слайд 2


Рисунок В.3 – Слайд 3

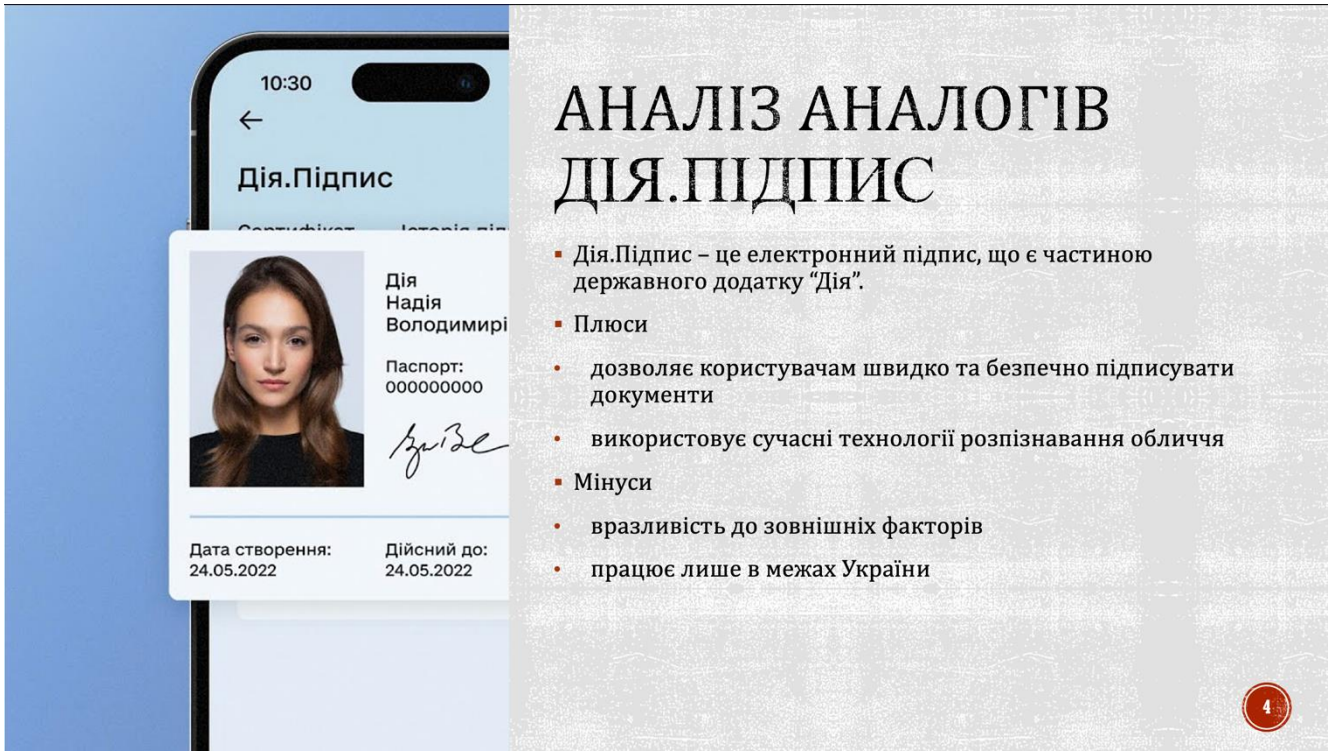



Рисунок В.4 – Слайд 4

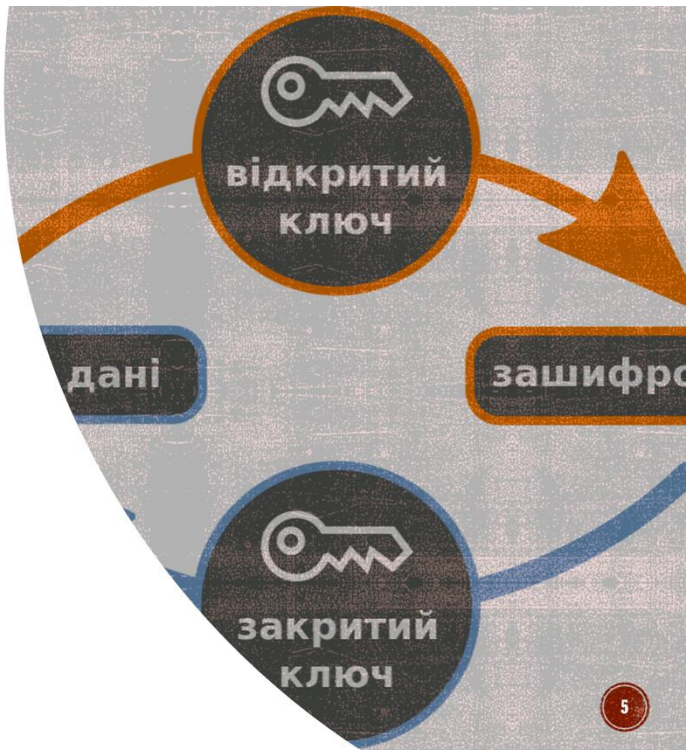



Рисунок В.5 – Слайд 5


Рисунок В.6 – Слайд 6

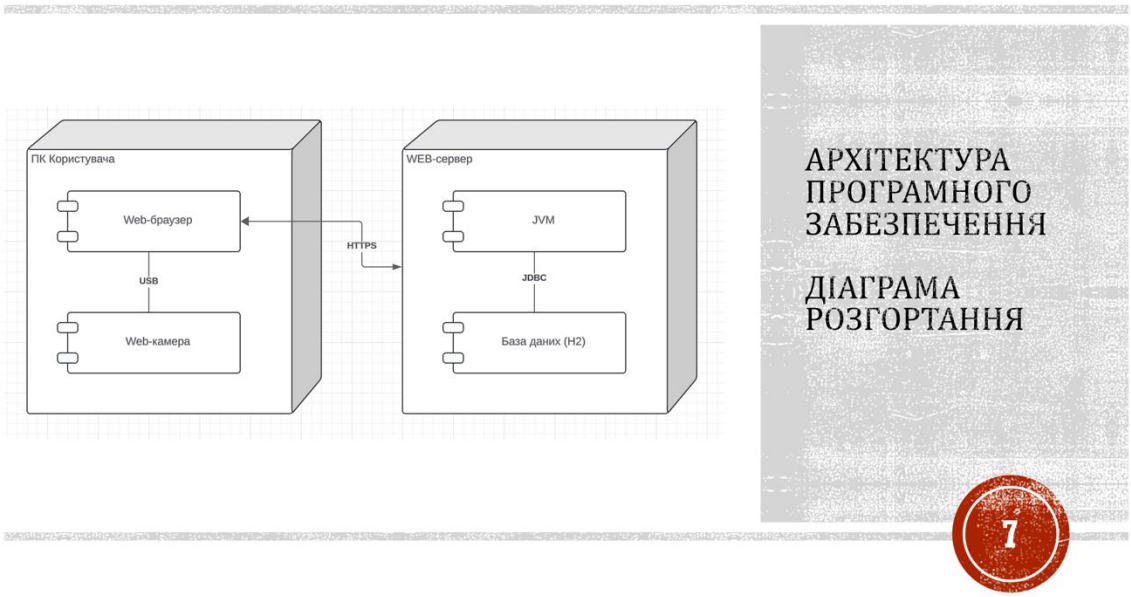



Рисунок В.7 – Слайд 7

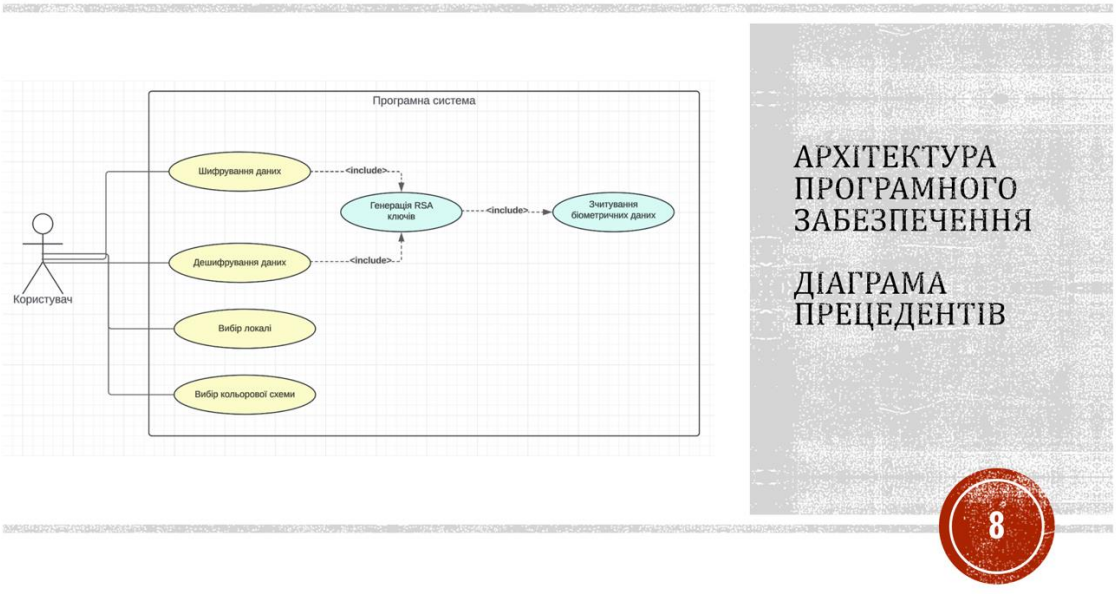



Рисунок В.8 – Слайд 8

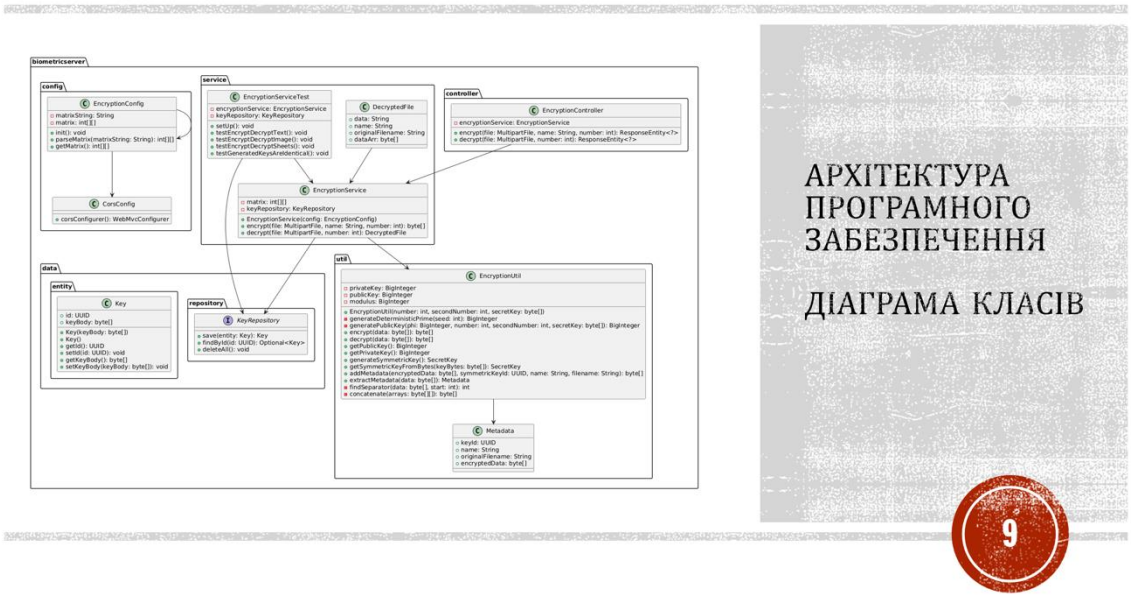



Рисунок В.9 – Слайд 9

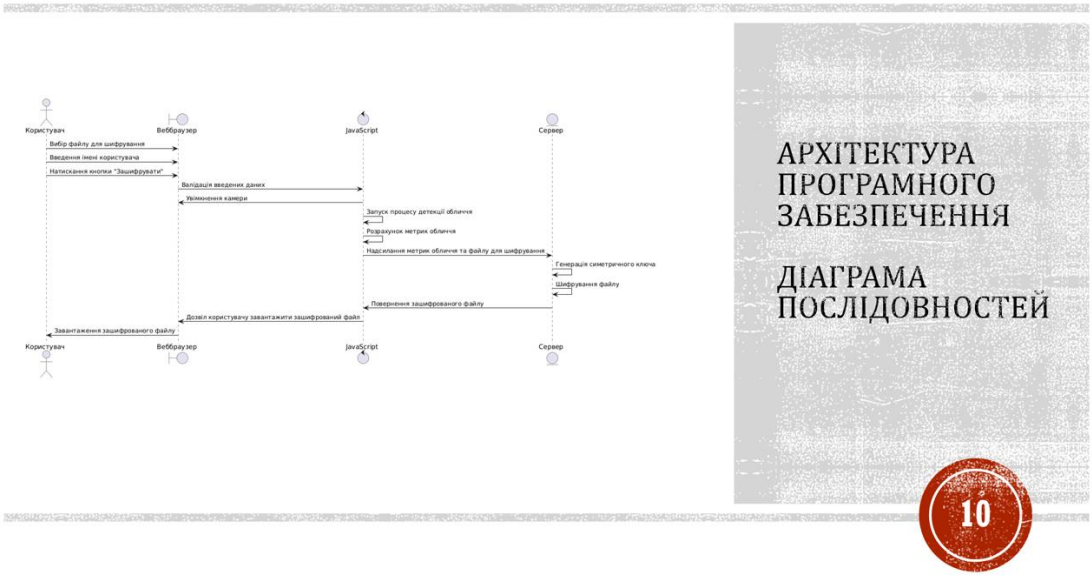



Рисунок В.10 – Слайд 10


Рисунок В.11 – Слайд 11


Рисунок В.12 – Слайд 12

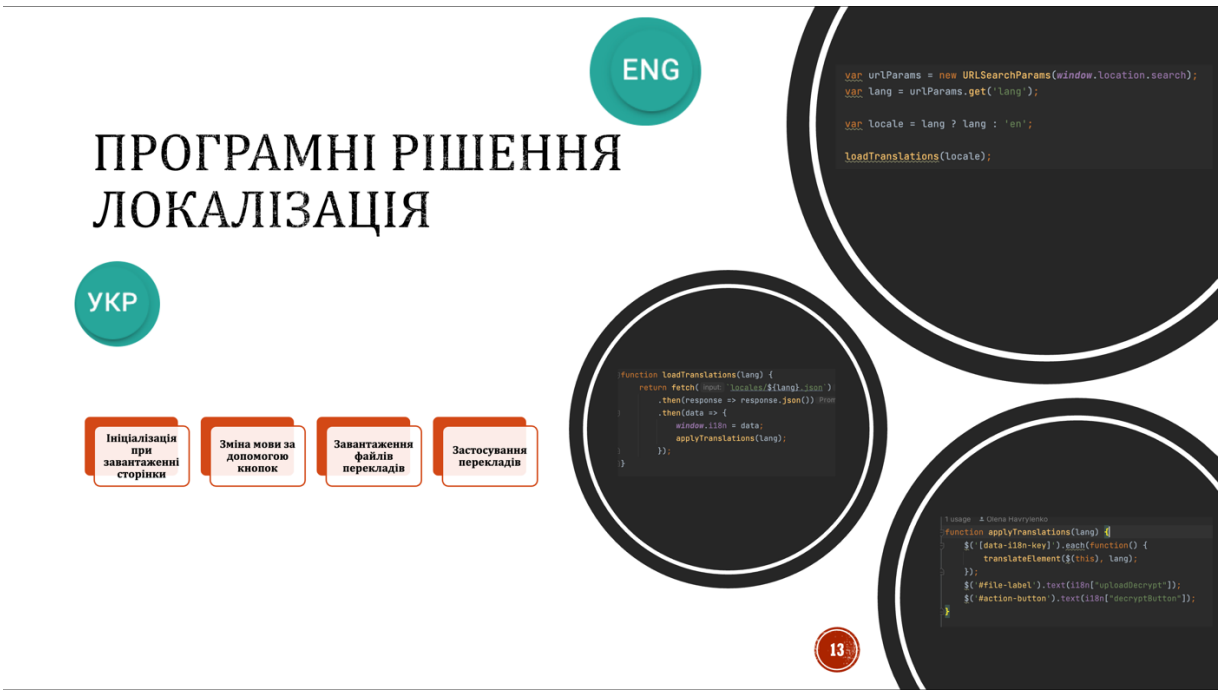



Рисунок В.13 – Слайд 13

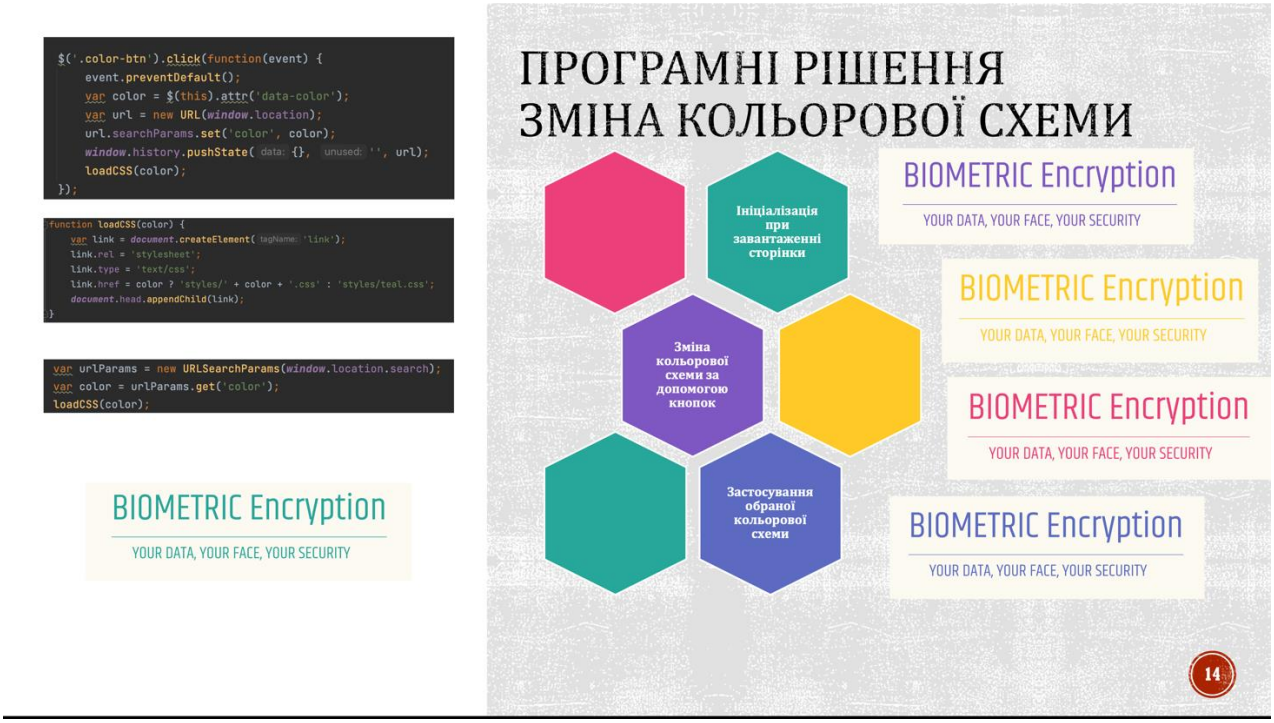



Рисунок В.14 – Слайд 14


Рисунок В.15 – Слайд 15

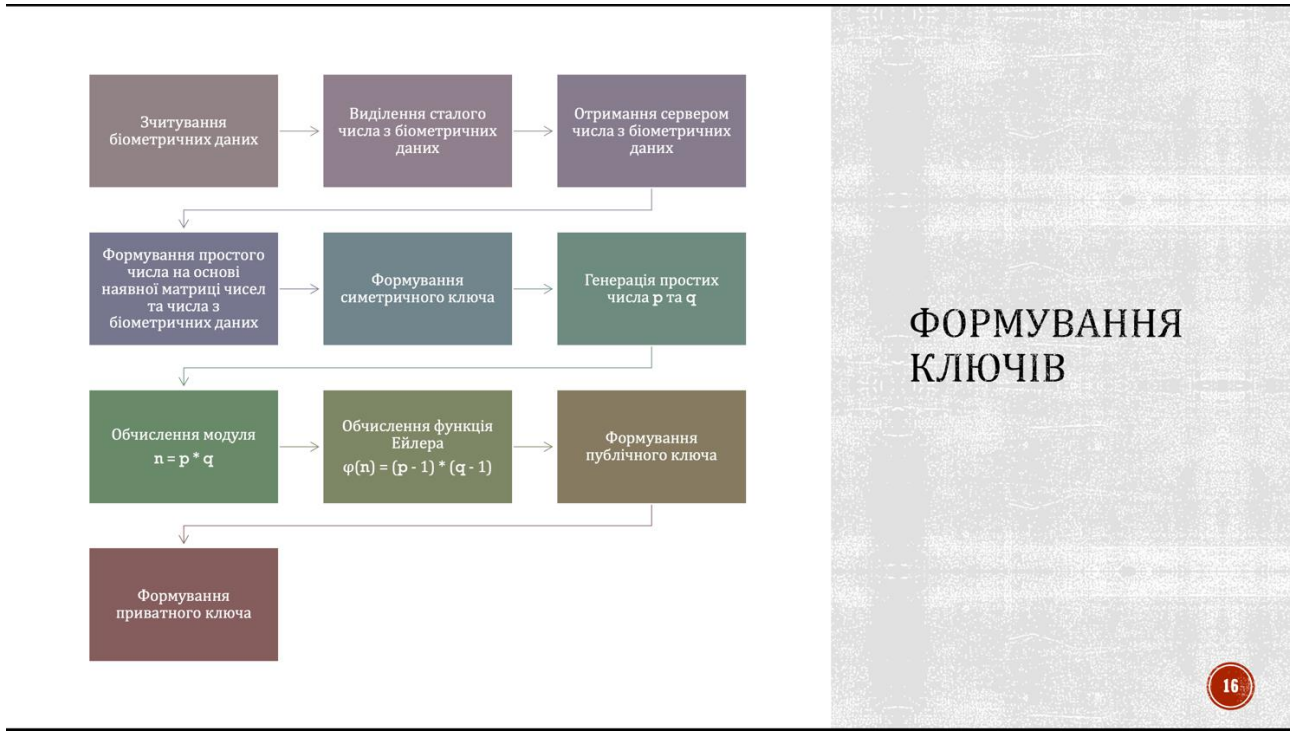



Рисунок В.16 – Слайд 16


Рисунок В.17 – Слайд 17


Рисунок В.18 – Слайд 18


Рисунок В.19 – Слайд 19



Рисунок В.20 – Слайд 20


Рисунок В.21 – Слайд 21


Рисунок В.22 – Слайд 22


Рисунок В.23 – Слайд 23


Рисунок В.24 – Слайд 24

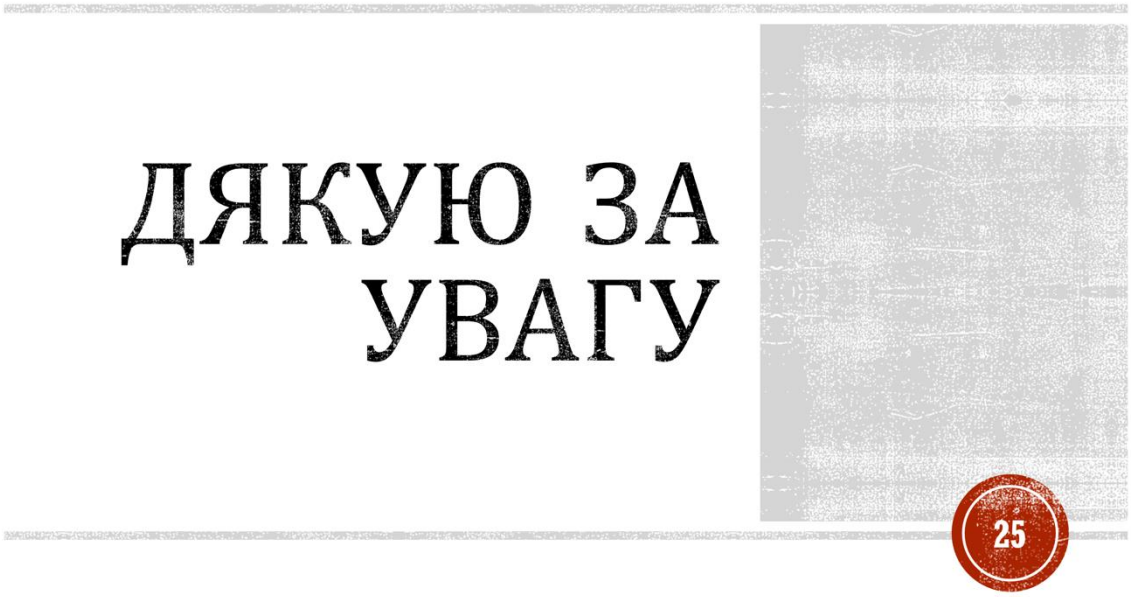



Рисунок В.25 – Слайд 25