Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет управління
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему: «Програмна реалізація системи ідентифікації та автентифікації з використанням криптоалгоритмів захищених каналів»

Виконав: студент групи ІПЗ20-1
Спеціальність
121 «Інженерія програмного забезпечення»
Ковальов Владислав Володимирович
(прізвище та ініціали)

Керівник д.т.н., професор кафедри комп'ютерних наук та інженерії програмного забезпечення
Яковенко В. О.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Університет митної справи та фінансів
(місце роботи)
в.о. завідувача кафедри Кібербезпеки та інформаційних технологій
(посада)
к. т. н., доцент кафедри кібербезпеки та інформаційних технологій Прокопович-Ткаченко Д. І.
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2024

АНОТАЦІЯ

Ковальов В.В. Програмна реалізація системи ідентифікації та автентифікації з використанням криптоалгоритмів захищених каналів.

Кваліфікаційна робота на здобуття освітнього ступеня бакалавр за спеціальністю 121 «Інженерія програмного забезпечення». – Університет митної справита фінансів, Дніпро, 2024.

Дана кваліфікаційна робота присвячена програмній реалізації системи ідентифікації та автентифікації з використанням криптоалгоритмів захищених каналів. В умовах стрімкого розвитку цифрових технологій та збільшення кількості додатків, захист персональної інформації стає все більш важливим. Ідентифікація та автентифікація є ключовими компонентами для забезпечення безпеки та конфіденційності даних користувачів. Вони дозволяють гарантувати, що доступ до додатку та його функцій мають лише авторизовані користувачі, що знижує ризик витоку даних та забезпечує захист конфіденційної інформації.

Наявність авторизованих користувачів є ключовим елементом у програмних системах безпеки, забезпечуючи надання конкретних прав доступу на основі ролі користувача в системі. Ідентифікація користувачів також дозволяє додаткам надавати персоналізовані послуги, враховуючи уподобання та історію використання конкретного користувача. У багатьох сферах, таких як фінанси, медицина та галузі, пов'язані з особистими даними, вимагається дотримання високих стандартів безпеки.

Робота досліджує використання криптографічних алгоритмів та захищених каналів передачі даних для забезпечення конфіденційності та цілісності інформації. Такі алгоритми дозволяють зашифровувати дані, що передаються по мережі, допомагаючи уникнути їх перехоплення та захистити від несанкціонованого доступу. Це дослідження спрямоване на розробку ефективних рішень для ідентифікації та автентифікації, що відповідають сучасним стандартам безпеки в умовах зростаючих загроз у кіберпросторі.

Ключові слова: HTTPS, SSL, TLS, AES, RAS, REST API, локальний сервер, публічний сервер.

ABSTRACT

Kovalov V.V. Software implementation of the identification and authentication system using crypto-algorithms of secure channels.

Qualification work for a bachelor's degree in speciality 122 «Software engineer». – University of Customs and Finance, Dnipro, 2024.

This qualification of the robot is dedicated to the software implementation of the identification and authentication system using various cryptoalgorithms for theft channels. In the minds of the rapid development of digital technologies and the increasing number of accessories, the protection of personal information is becoming increasingly important. Identification and authentication are key components to ensure the security and confidentiality of customer data. They can ensure that access to add-ons and functions is restricted to authorized users, which reduces the risk of data flow and ensures the protection of confidential information.

The visibility of authorized contributors is a key element in software security systems, ensuring that specific access rights are assigned based on the contributor's role in the system. Identification of clients also allows users to provide personalized services based on the medical history of a particular client. Many areas, such as finance, medicine and healthcare, which have special data requirements, require high safety standards.

The work monitors the use of cryptographic algorithms and secure data transmission channels to ensure confidentiality and integrity of information. Such algorithms allow you to encrypt data that is transmitted over the network, helping to prevent its storage and protect it from unauthorized access. This research is aimed at developing effective solutions for identification and authentication that meet current security standards in the face of growing threats in the cyberspace.

Keywords: HTTPS, SSL, TLS, AES, RAS, REST API, local server, public server.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AES – Advanced Encryption Standard

API – Application Programming Interface

CA – Certificate Authority

CAA – Certification Authority Authorization

CT – Certificate Transparency

FTP – File Transfer Protocol

GraphQL – Graph Query Language

HTTPS – Hypertext Transfer Protocol Secure

IP – Internet Protocol

JPA – Java Persistence API

JSON – JavaScript Object Notation

JWT – JSON Web Token

RAM – Random Access Memory

REST – Representational State Transfer

SHA – Secure Hash Algorithm

SOAP – Simple Object Access Protocol

SQL – Structured Query Language

SSH – Secure Shell

SSL – Secure Sockets Layer

SSD – Solid State Drive

TLS – Transport Layer Security

VDS – Virtual Dedicated Server

VNC – Virtual Network Computing

ПЗ – Програмне Забезпечення

ВСТУП

Актуальність дослідження. В умовах стрімкого розвитку цифрових технологій та збільшення кількості додатків, захист персональної інформації стає все більш важливим. Актуальність ідентифікації є ключовими компонентами для забезпечення безпеки та конфіденційності даних користувачів. Жодний додаток який потребує ідентифікувати користувачів не може обійтися без створення і розробки програмного забезпечення яке дозволить визначити конкретного користувача і його доступність до використання додатку.

Ідентифікація та автентифікація дозволяють гарантувати, що доступ до додатку та його функцій мають лише авторизовані користувачі. Це знижує ризик витоку даних та забезпечує захист конфіденційної інформації. Наприклад, мобільні додатки часто стають ціллю зловмисників. Використання надійних методів автентифікації значно ускладнює несанкціонований доступ і дозволяє забезпечити безпеку користувачу.

Наявність авторизованих користувачів є ключовим елементом у програмних системах безпеки. Ризик витоку даних та захист інформації забезпечується наданням конкретних права доступу на основі його ролі в системі. Це, в свою чергу, означає, що лише уповноважені особи можуть отримати доступ до конфіденційної інформації, що значно знижує ризик несанкціонованого доступу.

Ідентифікація користувачів дозволяє додаткам надавати персоналізовані послуги, враховуючи уподобання та історію використання конкретного користувача. Це підвищує зручність та задоволеність користувачів.

У багатьох сферах, таких як фінанси, медицина, та галузі, пов'язані з особистими даними, вимагається дотримання високих стандартів безпеки. Використання криптографічних алгоритмів захищених каналах дозволяє відповісти цим вимогам і захистити конфіденційні дані.

Для запровадження безпеки передачі даних, важливою частиною реалізації системи ідентифікації та автентифікації є використання криптоалгоритмів

захищених каналів передачі даних. Такі алгоритми дозволяють зашифровувати дані, що передаються по мережі, забезпечуючи їхню конфіденційність та цілісність. Це допомагає уникнути перехоплення інформації атакуючими та захистити дані від несанкціонованого доступу. При перехоплені даних захищених криптоалгоритмами, зашифровані дані залишатимуться недоступними без відповідного ключа.

Захист даних та безпека користувачів в цифрових додатках є одними з головних пріоритетів сучасного програмного забезпечення. Використання багатофакторної автентифікації, регулярне оновлення систем безпеки та впровадження передових методів шифрування забезпечують комплексний захист інформації. Враховуючи зростаючі загрози в кіберпросторі, важливо впроваджувати ефективні рішення для ідентифікації та автентифікації, які відповідатимуть сучасним стандартам безпеки.

Метою роботи є створення системи забезпечення віддаленої комунікації з використанням криптоалгоритмів захищених каналів.

Методи дослідження: аналіз літератури, обробка та аналіз інформації, методи створення архітектури багаторівневих додатків, дослідження сервісів по розгортанню додатків у глобальній мережі, планування та аналіз вимог системи, прототипування, методи проектування реляційних баз даних.

Об'єктом дослідження є процес передачі даних необхідних для ідентифікації та атентифікації, з використанням криптоалгоритмів захищених каналів.

Предметом дослідження є програмна система забезпечення ідентифікації та автентифікації для передачі даних та встановлення доступності функціоналу.

Практичне значення одержаних результатів – підвищення безпеки передачі даних та доступності функціоналу програмного забезпечення. Спрощення можливості передачі даних.

Структура роботи:

Розділ 1. Аналіз предметної області. Постановка задачі

Розділ 2. Аналіз і вибір методів реалізації та побудови ПЗ.

Розділ 3. Вирішення задачі реалізації системи ідентифікації та автентифікації з використанням криптоалгоритмів захищених каналів.

Робота складається зі вступу, 3-х розділів, висновків, списку використаних джерел з 20 найменувань. Обсяг роботи 53 сторінки, 12 рисунків.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ. ПОСТАНОВКА ЗАДАЧІ

1.1. Проблематика і актуальність дослідження

У сучасному мережевому світі багато застосунків потребують безпеки, і криптографія є одним з основних інструментів для її забезпечення. Основні цілі криптографії, такі як конфіденційність даних, цілісність даних, автентифікація та невідворотність (відповідальність), можуть використовуватись для запобігання різним видам мережевих атак, включаючи підслуховування, підробку IP-адрес, захоплення з'єднань та втручання в дані. Алгоритми є лише будівельними блоками у криптографічних протоколах, і розробка цих протоколів є надзвичайно складним завданням.

Криптографам важко створювати протоколи, що протистоять усім відомим атакам, а середньому розробнику це вдається ще гірше. Наприклад, розробники часто намагаються захистити мережеві з'єднання, просто шифруючи дані перед їх відправкою та розшифровуючи після отримання. Така стратегія часто не забезпечує цілісність даних. У багатьох випадках зловмисники можуть втрутитися у дані і навіть відновити їх.

Навіть якщо протоколи добре спроектовані, помилки в їх реалізації є звичайним явищем.

Протоколи для безпечного зв'язку через незахищені середовища мають широке застосування у більшості програмному забезпеченні. Основна мета протоколу SSL та його нащадка TLS (коли ми говоримо про SSL, маємо на увазі як SSL, так і TLS) полягає в наданні найбільш поширених послуг безпеки для довільних мережевих з'єднань на базі TCP таким чином, щоб мінімізувати потребу у криптографічній експертизі.

Основна мета криптографії – захист важливих даних, що передаються через середовище, яке саме по собі може бути незахищеним. Зазвичай таким середовищем є комп'ютерна мережа. Існує багато різних криптографічних алгоритмів, кожен з яких може забезпечувати одну або більше з наступних послуг для застосунків:

- конфіденційність (секретність): Дані залишаються секретними для тих, хто не має належних облікових даних, навіть якщо вони проходять через незахищене середовище. На практиці це означає, що потенційні зловмисники можуть бачити закодовані дані, але не зможуть їх розшифрувати без необхідної інформації. У класичній криптографії алгоритм шифрування був секретом. У сучасній криптографії алгоритми є публічними, і в процесах шифрування та дешифрування використовуються криптографічні ключі. Єдине, що потрібно зберігати в секреті, – це ключ;

- цілісність (захист від втручання): Основна ідея забезпечення цілісності даних полягає в тому, щоб одержувач міг визначити, чи були зроблені якісь зміни до даних протягом певного часу. Наприклад, перевірки цілісності можуть гарантувати, що дані, надіслані по каналу зв'язку, не були змінені під час передачі. Цілі класи алгоритмів шифрування піддаються атакам типу "bit-flipping", коли зловмисник може змінити значення біта даних, змінивши відповідний зашифрований біт;

- автентифікація: Криптографія може допомогти встановити особу для цілей автентифікації;

- невідворотність (відповідальність): Криптографія може дозволити довести, що отримане повідомлення, дійсно було надіслано відправником і не змінювалось. У реальному світі потрібно припускати, що зловмисник не скомпрометує певні криптографічні ключі. Протокол SSL не підтримує невідворотність, але це можна легко додати за допомогою цифрових підписів;

Ці прості послуги можуть бути використані для запобігання різним мережевим атакам, таким як:

- прослуховування (пасивне підслуховування): Зловмисник відстежує мережевий трафік та записує цікаві дані, наприклад, інформацію про кредитні картки;

- втручання: Зловмисник відстежує мережевий трафік і зловмисно змінює дані під час передачі (наприклад, може змінити вміст електронного листа).

- підробка: Зловмисник підробляє мережеві дані, роблячи вигляд, що вони надходять з іншої мережевої адреси. Така атака може бути використана для обману систем, що автентифікуються на основі інформації про хост (наприклад, IP-адреси);

- захоплення: Після автентифікації легітимного користувача підробка може бути використана для «захоплення» з'єднання;

- перехоплення та повторне відтворення: В деяких випадках зловмисник може записати та повторити мережеві транзакції з негативним ефектом. Наприклад, якщо ви продаєте одну акцію, коли ціна висока, зловмисник може записати цю транзакцію і повторити її пізніше, коли ціна акції впаде, що призведе до втрати всіх акцій сервісу [1].

1.2. Огляд існуючих рішень з методів розробки ідентифікації та автентифікації у ПЗ. Історія розвитку криптоалгоритмів захищених каналів управління

Вичерпна історія найважливіших подій, розглянута на ресурсі [2], які сформували екосистему криптоалгоритми захищених каналів:

- 1995, жовтень. Поява SSL v3;

SSL v3 (Secure Sockets Layer version 3) був випущений компанією Netscape і став першим широко використовуваним протоколом шифрування для забезпечення безпеки передачі даних через мережу Інтернет. Це був важливий крок у забезпеченні приватності та конфіденційності під час веб-сесій.

- 1995, листопад. Випуск PCT v1.0 від Microsoft;

PCT (Private Communication Technology) v1.0 був випущений Microsoft як альтернатива SSL v2. PCT був спроектований для покращення безпеки передачі даних, але не набув широкого поширення поза продуктами Microsoft.

- 1996, рік. Створення робочої групи IETF для стандартизації SSL;

У 1996 році була створена робоча група Internet Engineering Task Force (IETF) з метою стандартизації SSL, що веде до подальшого розвитку цього протоколу.

- 1999, січень. Опублікування TLS v1.0;

TLS (Transport Layer Security) v1.0 був випущений як наступна версія SSL v3 з покращеною безпекою та функціональністю. TLS v1.0 замінив SSL v3 у великій частині веб-комунікацій.

- 2006, квітень. Випуск TLS v1.1;

TLS v1.1 був випущений з метою виправлення певних уразливостей, які були виявлені в TLS v1.0. Цей протокол також впроваджував нові функції для підвищення безпеки.

- 2008, серпень. Випуск TLS v1.2;

TLS v1.2 став наступною ітерацією TLS з подальшими покращеннями безпеки та ефективності, включаючи виправлення деяких уразливостей, що були виявлені в попередніх версіях.

- 2011, березень. Публікація RFC 6176 для депрекації SSL v2;

RFC 6176 був опублікований з метою офіційного депрекування SSL v2 через велику кількість виявлених уразливостей та небезпек.

- 2012, березень. Заклик до пропозицій щодо HTTP/2;

Після успіху протоколу SPDY від Google, спільнота розробників виникла зацікавленість у розробці наступної ревізії протоколу HTTP, що веде до започаткування роботи над HTTP/2.

- 2013, березень. Запуск Certificate Transparency (CT) від Google.

CT створює публічний запис всіх публічних сертифікатів, що допомагає у виявленні недійсних або фальсифікованих сертифікатів, підвищуючи безпеку веб-комунікацій;

- 2013, серпень. Початок роботи над TLS 1.3;

Робота над TLS 1.3 почалася з метою створення більш безпечного та ефективного протоколу шифрування для мережевих з’єднань.

- 2014, січень. Перехід до RSA 2048 для сертифікатів;

Цей крок був зроблений для підвищення безпеки сертифікатів шляхом використання більш довгих ключів RSA.

- 2015, квітень. Випуск RFC 7469 для захисту від фальсифікації сертифікатів;

RFC 7469 ввів механізм публічного закріплення ключів для забезпечення додаткової захисту від фальсифікованих сертифікатів.

- 2015, травень. Реліз HTTP/2 як RFC 7540;

HTTP/2 був розроблений як наступна версія протоколу HTTP з метою покращення продуктивності та ефективності веб-комунікацій.

- 2016, лютень. Семінар "TLS 1.3 Ready or Not?";

Цей семінар став присвяченим аналізу та оцінці нових розробок у TLS 1.3.

- 2016, січень. Заборона видачі публічних SHA1 сер тифікатів;

У січні 2016 року було припинено видачу публічних сертифікатів з хешуванням SHA1, оскільки цей алгоритм вважався вже недостатньо безпечним для застосування в публічних сертифікатах.

- 2018, березень. Опублікування TLS 1.3 як RFC 8446;

TLS 1.3 був випущений з метою покращення безпеки та ефективності мережевих з'єднань. Він впроваджує нові функції, які дозволяють зменшити затримки та забезпечують більш високий рівень безпеки.

- 2018, липень. Позначення Chrome всіх HTTP-сторінок як "незахищені";

Google Chrome почав позначати всі сторінки, які використовують HTTP, як незахищені. Це стало однією з ініціатив для підвищення безпеки веб-переглядачів та заохочення використання протоколу HTTPS.

- 2023, грудень. Видалення Chrome іконки замка для шифрованого зв'язку.

Chrome припинив використання іконки замка, яка відображалася у веб-браузері для позначення шифрованого зв'язку, що стало кінцем певної ери та відображенням змін у підходах до візуалізації безпеки в мережі;

- 2023, жовтень. Додання підтримки S/MIME сертифікатів у RFC 9495;

RFC 9495 розширив можливості CAA (Certification Authority Authorization) до підтримки S/MIME сертифікатів, що покращило безпеку електронної пошти

та комунікацій з використанням S/MIME.2023, липень. Публікація Messaging Layer Security (MLS) як RFC 9420.

MLS став новим стандартом для захисту мережевих комунікацій та впровадження наскрізного шифрування в месенджерах та інших додатках.

- 2023, вересень. Депрекація TLS 1.0 і TLS 1.1 за оголошенням Apple;

Apple оголосила про відмову від підтримки застарілих версій TLS 1.0 і TLS 1.1 на своїх платформах з метою покращення безпеки мережевих з'єднань.

- 2024, квітень. Оновлена політика CT;

- 2024, квітень. Chrome за замовчуванням починає використовувати HTTPS;

Починаючи з версії 90, браузер Google Chrome за замовчуванням використовує HTTPS для більшості навігаційних запитів, які не вказують протокол, тим самим підвищуючи безпеку користувачів і веб-сайтів.

- 2024, березень. Депрекація TLS 1.0 і 1.1 за оголошенням IETF;

Internet Engineering Task Force (IETF) опублікувала RFC 8996, офіційно депрекуючи старі версії TLS 1.0 і 1.1. Це було зроблено через виявлені уразливості, які можуть бути використані для атак на безпеку мережевих з'єднань.

- 2024, лютий. Видача мільярда сертифікатів від Let's Encrypt.

Let's Encrypt оголосила, що видала понад мільярд сертифікатів для 192 мільйонів веб-сайтів. Це свідчить про широке використання шифрування та підвищену увагу до кібербезпеки в Інтернеті.

Звідси, можна зазначити основні сучасні методи забезпечення безпеки при передачі даних:

- TLS (Transport Layer Security);

TLS забезпечує конфіденційність, цілісність та аутентифікацію даних, що передаються між клієнтом і сервером через захищений канал. TLS використовує сучасні криптографічні алгоритми для шифрування даних, такі як AES (Advanced Encryption Standard), асиметричне шифрування на основі RSA або ECDSA, хеш-функції SHA (Secure Hash Algorithm) і багато інших.

- HTTPS (Hypertext Transfer Protocol Secure);

HTTPS забезпечує безпечну передачу даних між клієнтом і сервером за допомогою захищеного TLS-каналу. Він позначається у веб-браузерах за допомогою зеленого замка або іншого індикатора безпеки. HTTPS дозволяє захищати конфіденційні дані, такі як паролі, платіжні дані та особисту інформацію, від незаконного доступу та перехоплення.

- криптографічні алгоритми;

Симетричне шифрування: Використовується для шифрування даних під час передачі. Популярні алгоритми включають AES (Advanced Encryption Standard) і ChaCha20.

Асиметричне шифрування: Використовується для обміну ключами і аутентифікації. Прикладами є RSA (Rivest-Shamir-Adleman) і ECDSA (Elliptic Curve Digital Signature Algorithm).

Хешування: Використовується для створення унікальних хеш-кодів даних. Примерами є SHA-256 і SHA-3.

- Certificate Transparency (CT);

CT допомагає у виявленні недійсних або фальсифікованих сертифікатів, що забезпечує додатковий рівень безпеки у веб-комунікаціях. Сертифікати віддаються для реєстрації в публічний журнал, де їх можна перевірити на валідність та легітимність.

Ці методи і технології створюють надійний механізм для захисту даних під час їх передачі через Інтернет, що дозволяє забезпечити конфіденційність, цілісність та аутентичність інформації.

1.3. Висновок до першого розділу. Постановка задачі

Забезпечення безпеки передачі даних у сучасному мережевому середовищі є критично важливим завданням. Криптографія відіграє ключову роль у вирішенні цього завдання, оскільки вона забезпечує такі основні послуги, як конфіденційність, цілісність, автентифікацію та невідворотність.

Однак просте шифрування даних перед передачею та розшифрування після отримання не завжди є достатнім для забезпечення цілісності даних та захисту від різних видів мережевих атак. Розробка надійних криптографічних протоколів, які здатні протистояти відомим атакам, є надзвичайно складним завданням і часто призводить до помилок у реалізації.

Протоколи SSL/TLS (Secure Sockets Layer/Transport Layer Security) та HTTPS (Hypertext Transfer Protocol Secure) були розроблені з метою надання поширених послуг безпеки для мережевих з'єднань на основі TCP, мінімізуючи потребу в криптографічній експертизі. Їх розвиток та вдосконалення відбувалося протягом багатьох років, і вони стали de-facto стандартами для захищеного обміну даними в Інтернеті.

Крім того, впровадження таких технологій, як Certificate Transparency (CT), допомагає виявляти недійсні або фальсифіковані сертифікати, забезпечуючи додатковий рівень безпеки веб-комунікацій.

Використання сучасних криптографічних алгоритмів, таких як AES, ChaCha20, RSA, ECDSA, SHA-256 та SHA-3, разом із захищеними протоколами, такими як TLS та HTTPS, створює надійний механізм для захисту конфіденційності, цілісності та автентичності даних під час їх передачі через незахищені мережі, такі як Інтернет.

Виходячи з усього переліченого, можна сформулювати завдання, яке полягає у розробці програмної системи ідентифікації та автентифікації з використанням сучасних криптоалгоритмів для захищених каналів передачі даних: TLS (Transport Layer Security) та HTTPS (Hypertext Transfer Protocol Secure).

РОЗДІЛ 2. АНАЛІЗ І ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ ТА ПОБУДОВИ ПЗ

2.1. Обґрунтування оптимального варіанта реалізації завдання кваліфікаційної роботи

Для програмної реалізація системи ідентифікації та автентифікації з використанням криптоалгоритмів захищених каналів слід розглядати сучасні методи розробки, і орієнтуватися на останні стандарти безпеки, як було визначено постанові завдання у висновках до першого розділу.

Доцільно реалізувати систему на основі програмного забезпечення яке буде містити не систему ідентифікації та автентифікації, але й інший функціонал який буде доступний при успішності проходження автентифікації. Таким чином, можна буде наочно розглянути спільну роботу функціоналу додатку і системи ведення користувачів.

Тематику для основного функціоналу додатку було обрано віддалену систему зв’язку між комп’ютером та мобільним пристроєм для обміну файлами через локальний сервер. Такий функціонал може оптимізувати комунікацію для обміну даних і спростити взаємодію між пристроями.

Слід сформувати вимоги до програмного забезпечення для того щоб можна було оцінити оптимальність методів реалізації і закласти базу для розробки.

HTTPS вебсайти та програмного забезпечення використовує CT(Certificate Transparency), що використовується для передачі трафіку. CT включає інформацію про особу, яка пройшла процес перевірки ідентифікації, стандартизований Форумом браузерів CA у його базових вимогах. Цей процес підтверджує, що ідентифікований власник вебсайту, для якого видано сертифікат, має виключні права на використання домену [3].

SSL/TLS, вимагає використання надійного посередника. Цей надійний посередник цифрово підписує публічний ключ сервера — використовуючи алгоритми, криптоалгоритмів — і клієнт повинен перевірити цей підпис. Такий підписаний публічний ключ називається сертифікатом, а надійний посередник, відповідальний за підписання сертифікатів, називається центром сертифікації

(CA). Клієнт повинен мати доступ до публічного ключа CA, щоб аутентифікувати підпис перед прийняттям ключа як справжнього. Веб-браузери мають список довірених CA з їхніми публічними ключами, вбудованими саме для цієї мети.

Потрібна розробка способу асоціювати публічний ключ з сервером, до якого підключається додаток. Тому належним чином оформлений сертифікат повинен містити не тільки публічний ключ сервера, але й доменне ім'я сервера, до якого належить публічний ключ, все це підписане довіреним посередником [4].

Вищеперелічене дає змогу побудувати основну модель системи реалізації завдання. Модель відтворена на рис. 2.1.

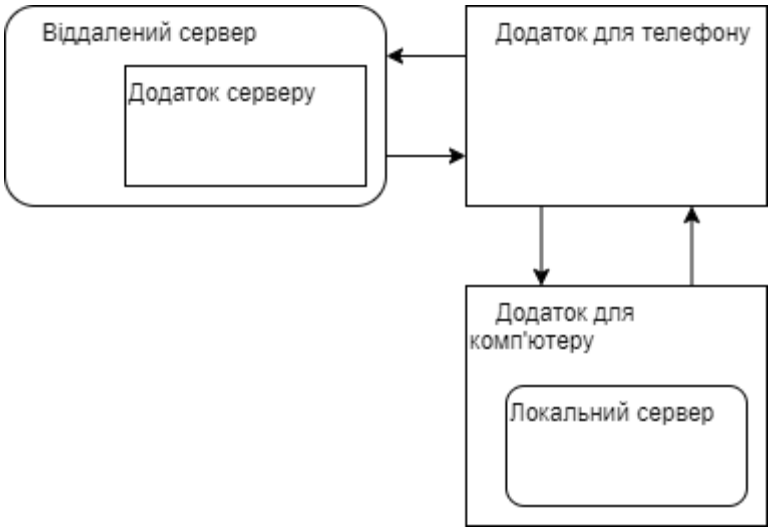

Рисунок 2.1 – Основна модель системи завдання з найвищем рівнем абстракції

Для щоб отримати СТ існують наступні способи отримання:

- отримати сертифікат від довіреного CA який буде ним підписаний;
- згенерувати самопідписаний сертифікат.

Порівняння способів отримання сертифікату:

Перший спосіб потребує покупки доменного імені для віддаленого серверу для того щоб отримати довірений СТ(наприклад Let`s Encrypt) [5]. При цьому

підключення до віддаленого серверу через протокол HTTPS буде можливо через будь який додаток клієнту, браузер чи наприклад інструменти тестування API, як Postman [6].

Другий спосіб також дає змогу реалізувати підключення по протоколу HTTPS, але потребує наступних факторів:

- генерація самопідписаного СТ;

- для генерації не важливо мати доменне ім’я для серверу, достатньо його IP адреси;

- окрім підключення до програмного забезпечення яке знаходиться на віддаленому сервері, підключити до клієнтсього додатку, щоб операційна система визнавала згенерований СТ як довірений.

Орієнтуючись на порівняння було обрано другий варіант через те що він не потребує додаткових матеріальних витрат, дозволить більше поглибитись у роботу протоколу HTTPS та підключення СТ до серверу.

Для створення додатку на віддаленому веб-сервері потрібно проаналізувати та обрати модель для побудови веб-сервісів та API.

Існують наступні моделі для побудови веб-сервісів та API:

- SOAP (Simple Object Access Protocol) є протоколом обміну структурованими повідомленнями в мережі. Його основна особливість полягає в тому, що він базується на XML для передачі даних та має строгу специфікацію стандартів. SOAP зазвичай використовується для взаємодії між різними програмними компонентами через мережу, такими як веб-сервіси. Він використовує HTTP або інші протоколи для передачі повідомлень, а структуровані дані представляються у вигляді XML [7];

- GraphQL - це мова запитів для API, яка дозволяє клієнтам визначати, які саме дані їм потрібні з однієї точки входу. Вона надає більш гнучкий та ефективний спосіб отримання даних, оскільки клієнти можуть запитувати лише ті поля, які їм необхідні, у зручному для них форматі. GraphQL також дозволяє виконувати складні запити та отримувати зворотні дані в одному запиті [8];

- gRPC (Google Remote Procedure Call) - це сучасна технологія RPC (віддаленого виклику процедур), розроблена Google. Вона використовує бінарний протокол для передачі даних, що дозволяє зменшити розмір повідомлень та покращити продуктивність передачі даних. gRPC дозволяє легко визначати служби та їх методи за допомогою файлів з описом інтерфейсів, що спрощує розробку та робить код більш читабельним [9];

- WebSockets - це протокол двостороннього зв'язку поверх TCP, який забезпечує повний дуплексний зв'язок між клієнтом і сервером. Його використання дозволяє побудувати інтерактивні веб-додатки, які можуть обмінюватися даними в режимі реального часу. Наприклад, веб-чати, онлайн-ігри або веб-приложення з потоковою передачею даних. WebSockets дозволяють ефективно використовувати серверні ресурси та підтримувати активне з'єднання між клієнтом і сервером без необхідності постійної повторної ініціалізації з'єднання [10];

- REST API (Representational State Transfer) - це архітектурний стиль для створення веб-сервісів, який використовує HTTP-протокол для передачі даних і операцій. REST API використовує стандартні HTTP-методи, такі як GET, POST, PUT, DELETE, для виконання операцій з ресурсами сервера. REST API використовує ресурси (URL) і їхні стани (стан ресурсу) для взаємодії з клієнтами [11].

REST API зазвичай вважається найкращим вибором з кількох причин:

- REST API використовує звичайні HTTP-методи (GET, POST, PUT, DELETE), що робить його інтуїтивно зрозумілим та легким у вивченні.

Завдяки безстановій архітектурі REST API дуже добре масштабується для обслуговування великої кількості клієнтів;

- REST API добре працює з різними форматами даних, такими як JSON або XML, і легко інтегрується з більшістю веб-технологій;

- відповіді REST API можна кешувати на стороні клієнта або на проміжних серверах, що покращує продуктивність;

- REST API не прив'язаний до конкретної мови програмування, що забезпечує гнучкість у виборі технологій.

Як зазначається у статті [12] є рекомендованим способом передачі даних автентифікації та авторизації між клієнтом і сервером у веб-додатках є JWT(JSON Web Tokens), що дасть змогу забезпечити більш велику та контрольовану безпеку.

Переваги використання:

- JWT - це компактний розмір даних у форматі JSON, що забезпечує ефективну передачу через HTTP-заголовки або URL-параметри;

- JWT можуть бути підписані за допомогою секретного ключа або пари публічного/приватного ключів, що забезпечує захист від підробки та гарантує цілісність даних;

- JWT не вимагає зберігати стан на сервері, оскільки вся необхідна інформація зберігається в самому токені, що робить систему більш масштабованою;

- JWT можуть бути використані на різних платформах і мовах програмування, оскільки базуються на відкритих стандартах;

- JWT можуть бути декодовані на стороні клієнта для отримання певної інформації, наприклад, про користувача, без додаткових запитів на сервер;

- JWT можуть містити довільні дані в зашифрованому форматі, що дозволяє передавати додаткову інформацію між клієнтом і сервером;

- JWT можуть бути кешовані на стороні клієнта або на проміжних серверах, покращуючи продуктивність;

- JWT є відкритим стандартом (RFC 7519), що сприяє широкому використанню та підтримці різними бібліотеками та фреймворками.

Завдяки своїй безпеці, масштабованості, гнучкості та відповідності стандартам, JWT став провідною технологією для передачі даних автентифікації та авторизації у сучасних веб-додатках.

Для основного функціоналу по обміну файлами через локальний сервер доцільним буде використання Socket, через те що він надає наступні переваги порівняно з іншими методами комунікації у мережевих додатках:

- низькорівневий доступ до мережевого рівня, дозволяючи розробникам здійснювати точне управління передачею даних, включаючи керування буферами, обробкою помилок, відновленням з'єднання та іншими аспектами мережевого взаємодії;

- дозволяє використовувати різні мережеві протоколи, такі як TCP, UDP, IP і т. д., Що робить його універсальним і гнучким для різних потреб мережевого програмування;

- забезпечує ефективну передачу даних, оскільки дозволяє використовувати буферизацію даних, оптимізацію передачі пакетів і інші методи для зменшення затримок і підвищення продуктивності;

- дозволяє реалізувати захист даних за допомогою шифрування, аутентифікації та інших методів захисту мережевої комунікації;

- Socket може бути використаний для реалізації асинхронної моделі програмування, що дозволяє обробляти багатозадачність і запити без блокування основного потоку виконання програми;

- надає розширені можливості налаштування і контролю мережевого з'єднання, такі як таймаути, розмір буферів, маршрутизація і т. д., Що дозволяє оптимізувати та керувати мережевою комунікацією;

- більшість мов програмування, включаючи C#, підтримують Socket, що робить його крос-платформенним і дозволяє використовувати один і той же код для різних операційних систем і платформ.

Усі ці переваги роблять Socket потужним інструментом для розробки мережевих додатків, особливо коли потрібно точне управління мережевою взаємодією і висока продуктивні

2.2. Висновок до першого розділу. Постановка задачі

У другому розділі розглянуто програмну реалізація системи ідентифікації та автентифікації з використанням криптоалгоритмів захищених каналів. Основними аспектами є орієнтація на останні стандарти безпеки, реалізація системи з додатковим функціоналом для оптимального використання, і обрання моделі побудови веб-сервісів та API.

Для ефективної реалізації системи, розглянуте використання сучасних методів розробки та орієнтування на стандарти безпеки. Зазначено про важливість реалізації системи з додатковим функціоналом для вивчення спільної роботи функціоналу додатку і системи ведення користувачів.

Проаналізовано використання HTTPS для забезпечення безпеки трафіку з урахування Certificate Transparency (CT) та SSL/TLS для аутентифікації сервера.

Описаний і аргументований виборі моделі побудови веб-сервісів та API. Обрано модель REST арі через його простоту, гнучкість, масштабованість та незалежність від мов програмування. Для передачі даних автентифікації та авторизації зазначено про доцільність використовування JWT, що забезпечить більш велику та контрольовану безпеку.

РОЗДІЛ 3. ВИРІШЕННЯ ЗАДАЧІ РЕАЛІЗАЦІЇ СИСТЕМИ ІДЕНТИФІКАЦІЇ ТА АВТЕНТИФІКАЦІЇ З ВИКОРИСТАННЯМ КРИПТОАЛГОРИТМІВ ЗАХИЩЕНИХ КАНАЛІВ

3.1. Опис програмного забезпечення і розробка архітектури додатку веб-серверу

Для створення програмного забезпечення яке буде працювати захищеними каналами управління та дозволить проводити ідентифікацію та автентифікацію було обрано мову програмування Kotlin і фреймворки Spring, Spring Security. Це дозволило розробити гнучкий і безпечний веб-сервер.

Spring Security надає рішення для управління автентифікацією, авторизацією, захистом від атак та іншими аспектами безпеки. Він дозволяє налаштовувати рівні доступу, використовувати різні методи аутентифікації та забезпечує інтеграцію з іншими компонентами Spring. Також даний фреймворк використовує сучасні криптографічні алгоритми для збереження та обробки паролів, а також для забезпечення безпеки передачі даних.

Для збереження даних користувача обрано реляційну базу даних MySQL. MySQL є одним із найпоширеніших реляційних баз даних, відомих своєю надійністю та стабільністю, що робить її відмінним вибором для систем, які обробляють важливі дані користувачів. Вона підтримує можливість масштабування, дозволяючи розширювати обсяг даних і кількість одночасних з’єднань для обслуговування більшої кількості користувачів. Оптимізований двигун бази даних забезпечує добру швидкодію, що особливо важливо для веб-серверів, які обробляють великі обсяги запитів від користувачів. MySQL надає механізми безпеки, такі як аутентифікація та авторизація користувачів, що дозволяють захищати доступ до даних і запобігати несанкціонованому доступу.

Для оптимізації, полегшення і прискорення впровадження бази даних було використано JPA(Java Persistence API). Використання JPA для ідентифікації та автентифікації користувачів через веб-сервер має кілька важливих переваг. По-перше, JPA забезпечує ORM (Object-Relational Mapping), що дозволяє легко

перетворювати об’єкти Java в записи бази даних і навпаки, спрощуючи роботу з базою даних і зменшуючи кількість коду. По-друге, оскільки JPA є стандартом, це дозволить переносити код між реалізаціями JPA за необхідності, такими як Hibernate або EclipseLink, що забезпечує легкість зміни баз даних без значних змін у коді. По-третє, JPA добре інтегрується з Spring Framework, зокрема Spring Security, що спрощує налаштування аутентифікації та авторизації користувачів.

Використовуючи Spring Data JPA, можна створювати репозиторії для керування користувачами з мінімальною кількістю коду. Крім того, JPA підтримує кешування даних, що покращує продуктивність програми, і надає механізми для оптимізації запитів, такі як жадне і ліниве завантаження асоціацій.

JPA забезпечує вбудовану підтримку транзакцій, що дозволяє підтримувати цілісність даних під час операцій створення, оновлення та видалення користувачів. Використання JPA допомагає зберігати консистентність даних між об’єктами Kotlin і базою даних. Крім того, JPA підтримує великі навантаження і може бути налаштована для роботи в кластеризованих середовищах, що робить її масштабованою.

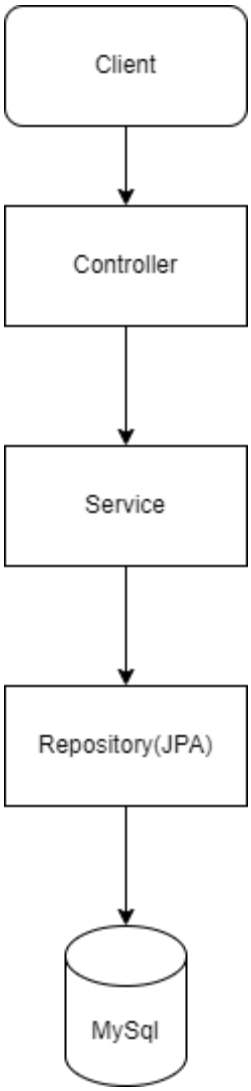

Рисунок 3.1 – Архітектура веб-серверу

На рис.3.1 показана загальна архітектура REST API веб-серверного додатку:

Client: програмний компонент який відправляє HTTPS-запити на веб-сервер з метою отримання ресурсів або виконання певних операцій над ресурсами. У розрізі даної роботи – це Android додаток.

Controller: компонент додатку який обробляє HTTPS-запити і визначає, які дії потрібно виконати у відповідь на ці запити. Контролер служить точкою входу для клієнтів REST API, зв’язуючи їх запити з відповідними методами, які реалізують бізнес-логіку у сервісному шарі архітектури.

Service: проміжний рівнем між контролерами (Controllers) і шаром доступу до даних (Repository). Основна мета шару сервісу полягає у виконанні бізнес-

логіки, обробці даних і координації різних операцій, які можуть включати взаємодію з кількома репозиторіями або зовнішніми сервісами.

Шар сервісу дозволяє розділити бізнес-логіку від логіки обробки HTTPS-запитів, що сприяє кращій структурованості коду, спрощує тестування і полегшує підтримку та розвиток додатку.

Repository(JPA): виступає в ролі шару надання і доступу даних. Цей шар відповідає за взаємодію з базою даних і виконання операцій з даними, такими як збереження, отримання, оновлення та видалення.

Основна мета шару Repository полягає у тому, щоб ізолювати бізнес-логіку від деталей роботи з базою даних. Це сприяє покращенню структурованості коду, підтримці розділення обов'язків.

MySQL: реляційна база даних яка використовується для збереження даних користувачів, доступність до них, SQL запити на отримання даних.

Розглянемо архітектуру Controllers представлену на рис. 3.2.

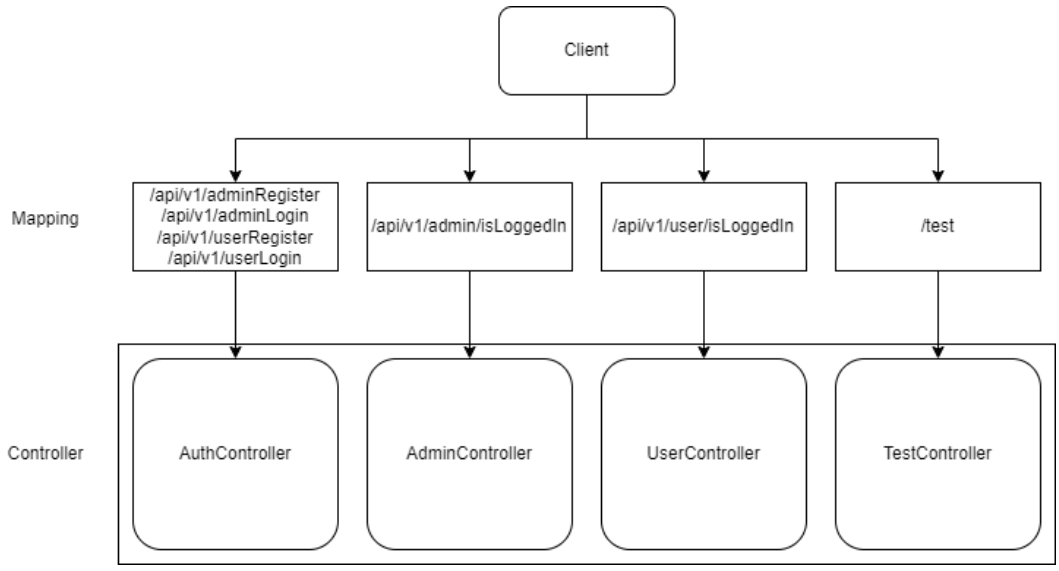

Рисунок 3.2 – Mapping доступу до шару Controller

Mapping – встановлює зв'язок між URL-шляхом та методом контролера, який буде виконуватися при запиті на цей URL-шлях веб-серверу.

Компоненти з рис.3.2:

AuthController: Controller який відповідає за Mapping для ідентифікації та втентифікації користувача.

Приклади формату даних для запитів і відповідей:

/api/v1/userLogin

Request POST json:

{
“email”: “test@gmail.com”,
“password”: “123456”
}

Response json:

{
“success”: true,
“message”: “login successful !!”,
“token”:
“eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0QgdtYWlsLmNvbSIsImlhdCI6MTcxNjkwMzYzMywiZXhwIjoxNzE2OTAzNzAzLCJ1c2VydHlwZSI6IlVTRVIifQ.cQw4SBsxfFEhlaFoBCbqT0PkkFW3vfq44ZlUh9WDKgQ”,
“user”: {
“username”: “user”,
“email”: “test@gmail.com”,
“id”: 1
}
}

/api/v1/userRegister

Request POST json:

{
“username”: “user”,
“email”: “test@gmail.com”,
“password”: “123456”
}

Response json:

{
“success”: false,
“message”: “Email is already registered !!”
}

/api/v1/adminLogin

Request POST json:

{
“username”: “admin123”,
“password”: “123456”
}

Response json:

{
“success”: true,
“message”: “login successful !!”,
“token”:
“eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0QgdtYWlsLmNvbSIsImlhdCI6MTcxNjkwMzYzMywiZXhwIjoxNzE2OTAzNzAzLCJ1c2VydHlwZSI6IlVTRVIifQ.cQw4SBsxfFEhlaFoBCbqT0PkkFW3vfq44ZlUh9WDKgQ”,
“user”: {
“username”: “user123”,
“id”: 1
}
}

/api/v1/userRegister

Request POST json:

{
“username”: “user123”,
“password”: “123456”
}

Response text:

“User Register successful !! “

AdminController: Controller який відповідає за Mapping для функціоналу передбачуваному лише користувачу з роллю Admin.

/api/v1/admin/isLoggedIn

Request GET:

Header:»Authorization»:»Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMTIzIiwiaWF0IjoxNzE3MTEyMjEzLCJleHAiOjE3MTcxMTIyODMsInVzZXJ0eXBlIjoiQURNSU4ifQ.wKQRcMoOoQt7AsZWgfqmdMgbfbrYK4HdtOxI7gztHXM»

Response:

«true»

UserController: Controller який відповідає за Mapping для функціоналу передбачуваному лише користувачу з роллю User.

/api/v1/user/isLoggedIn

Request GET:

Header:»Authorization»:»Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyMTIzIiwiaWF0IjoxNzE3MTEyMjEzLCJleHAiOjE3MTcxMTIyODMsInVzZXJ0eXBlIjoiQURNSU4ifQ.wKQRcMoOoQt7AsZWgfqmdMgbfbrYK4HdtOxI7gztHXM»

Response:

“true”

TestController: Controller який відповідає за Mapping для тестування веб-серверу.

/test

Request GET:

“test”

Розглянемо архітектуру Service + Repository представлену на рис. 3.3.

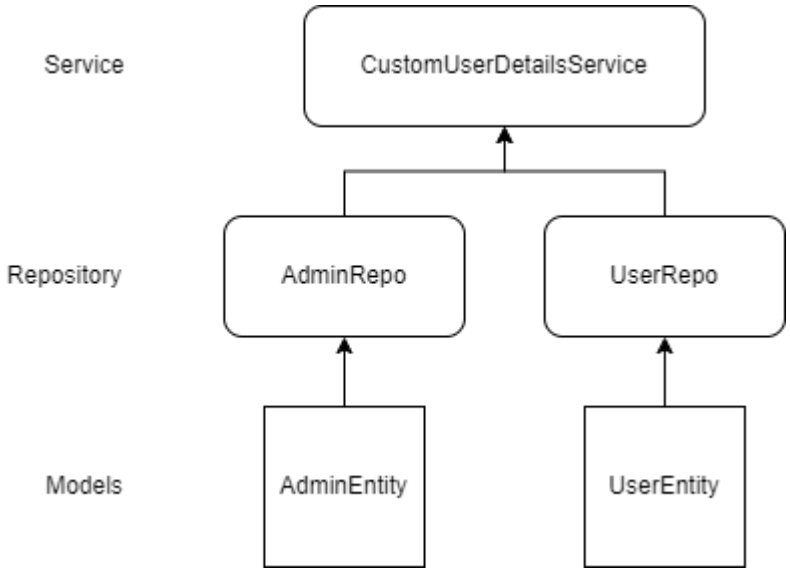

Рисунок 3.3 – Класи і сервіси взаємодії Service, Repository шарів архітектури

CustomUserDetailsService відповідає за реалізацію інтерфейсу UserDetailsService в Spring Security. Цей інтерфейс має метод loadUserByUsername, який викликається для завантаження користувача за його ім’ям користувача під час аутентифікації.

Метод loadUserByUsername виконує наступне:

- перевіряє, чи встановлено значення userType. Це значення представляє тип користувача (UserType), такий як ADMIN або USER;

- в залежності від типу користувача, метод виконує запит до відповідного репозиторію (adminRepo або userRepo) для отримання даних про користувача за ім’ям або електронною поштою;

- створює об’єкт UserDetails, який представляє користувача у системі Spring Security. Для цього використовуються класи SimpleGrantedAuthority для ролі користувача та User для представлення користувача;

- повертає об’єкт UserDetails;

- клас CustomUserDetailsService відповідає за обробку запитів на завантаження користувачів з бази даних та їх представлення у форматі, придатному для автентифікації і авторизації у системі Spring Security.

Шар Repository містить інтерфейси AdminRepo і UserRepo, які наслідують від JpaRepository.

AdminRepo:

findByUsername(username: String): Optional<AdminEntity>: Цей метод виконує запит до бази даних і повертає об’єкт AdminEntity за його ім’ям користувача (username). Якщо користувача з таким ім’ям не знайдено, повертається пустий Optional.

existsByUsername(username: String): Boolean: Цей метод перевіряє, чи існує користувач з вказаним ім’ям користувача (username) у базі даних. Повертає true, якщо користувач існує, і false в іншому випадку.

UserRepo:

findByEmail(email: String): Optional<UserEntity>: Цей метод аналогічний до findByUsername, але шукає користувача за його електронною адресою (email).

existsByEmail(email: String): Boolean: Цей метод аналогічний до existsByUsername, але перевіряє наявність користувача за електронною адресою (email).

Шар Models містить дві сутності (Entity) – AdminEntity і UserEntity, які представляють таблиці бази даних, і які використовуються для створення таблиць і сутностей у базі даних.

Для підключення СТ для забезпечення HTTPS з’єднання з веб-сервером налаштовується конфігурація у файлі проекту за шляхом src/main/resourses/application.yml як показано на рис. 3.4.

```yaml
server:
  port: 8181
  ssl:
    key-store: classpath:keystore.p12
    key-store-password: 123456
    key-store-type: PKCS12
    key-alias: mylocalcert

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/remote-control-db
    username: root
    password: 123456
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
    database-platform: org.hibernate.dialect.MySQL5Dialect
```

Рисунок 3.4 – Файл конфігурації веб-серверу

Опис основних параметрів конфігурації веб-сервера наведено у таблиці 3.1.

Таблиця 3.1

Опис основних параметрів конфігурації веб-сервера

Параметр	Значення
port	Вказує порт, на якому буде працювати веб-сервер. У цьому випадку, додаток буде доступний за адресою http://ip-серверу:8181/
key-store	Вказує шлях до файлу з ключами для SSL, у цьому випадку, keystore.p12
key-store-password	Пароль доступу до keystore
key-store-type	Тип keystore, у цьому випадку, PKCS12
key-alias	Псевдонім ключа в keystore
url	URL для доступу до бази даних MySQL
driver-class-name	Назва драйвера JDBC для взаємодії з базою даних.
Ddl-auto	Режим автоматичного створення / оновлення таблиць бази даних на основі сутностей JPA

Для того щоб згенерувати самопідписаний СТ, потрібно використати команду у терміналі операційної системи:

keytool -genkeypair -alias mylocalcert -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore keystore.p12 -validity 3650 -ext SAN=IP:127.0.0.1

При цьому у параметр SAN потрібно змінити IP-адресу 127.0.0.1 на IP-адресу зовнішнього веб-серверу який використовується для хостингу веб-серверу для ідентифікації та автентифікації користувачів.

Перевірити згенерований SSL сертифікат можна наступною командою [13], яку відтворено на рис. 3.5:

keytool -list -keystore keystore.p12 -storetype PKCS12

```
PS C:\Projeckts\Kotlin\diplom server\remote-control-server\src\main\resources> keytool -list -k
eystore keystore.p12 -storetype PKCS12
Enter keystore password:
Keystore type: PKCS12
Keystore provider: SUN
Your keystore contains 1 entry
mylocalcert, 21 мая 2024 г., PrivateKeyEntry,
Certificate fingerprint (SHA-256): 0B:C2:AA:47:0D:42:01:3C:2D:E8:8F:D5:6C:60:E7:1B:9A:E7:46:A2:
C4:A3:CA:02:DB:DA:61:6C:FF:BB:68:31
```

Рисунок 3.5 – SHA-256 згенерованого сертифікату з прикладу

Згенерований сертифікат потрібно розмістити у проекті на шляху src/main/resourses/

Як результат створення веб-серверу можна отримати веб-сервер по ідентифікації і автентифікації користувачів, із використанням захищених каналів управління HTTPS, SSL/TLS, та який використовує наступні криптографічні алгоритми:

- алгоритм обміну ключами: RSA;

Використання алгоритму RSA вказується під час генерування сертифіката командою за допомогою опції -keyalg RSA.

Розмір ключа RSA: 2048 біт

У команді створення сертифіката вказано -keysize 2048, що означає використання ключа RSA розміром 2048 біт.

- алгоритм симетричного шифрування: AES.

У конфігурації application.yml не вказано конкретний алгоритм симетричного шифрування і його розмір, тому використовуватиметься алгоритм за умовчанням для TLS Spring. За замовчуванням Spring використовує алгоритм AES для симетричного шифрування TLS.

Розмір ключа AES: 256 біт

3.2. Опис апаратного і програмного забезпечення для створення зовнішнього серверу

Для того щоб реалізувати сервер з моделлю REST API по ідентифікації користувачів було використано сервіс Friend Hosting і куплено Progressive SSD VDS сервер по тарифу Micro NVMe, який має наступні характеристики:

vCPU: 1 core - Це означає, що у вас є одне віртуальне процесорне ядро для обробки завдань на сервері. Це може бути достатньо для початкового етапу розробки та тестування, але може вимагати масштабування при збільшенні навантаження.

RAM: 512 MB - Обсяг оперативної пам'яті на сервері. Для невеликих додатків цього може бути достатньо, але для більш складних систем, особливо з багатьма користувачами, потрібно буде розглядати збільшення обсягу пам'яті.

Disk: 10 GB NVMe - Обсяг дискового простору для зберігання файлів, баз даних та іншої інформації на сервері. 10 ГБ може бути достатньо для початкової розробки, але з часом може знадобитися розширення, особливо якщо ви зберігаєте велику кількість даних.

IPv4: 1 шт. - Одна IPv4-адреса для доступу до вашого сервера через Інтернет. Це необхідно для забезпечення зв'язку з вашим додатком.

Port: до 1 Gbps - Це швидкість мережі, яка вимірюється в гігабітах на секунду. Цей тариф дозволяє вам використовувати порти зі швидкістю до 1 Гбіт/с, що забезпечує швидкий доступ до вашого додатку через мережу.

Тип твердотілий накопичувач для пам’яті: SSD

Також слід зазначити що сервіс Friend Hosting надає підтримку та можливість масштабування сервера в майбутньому, якщо це буде необхідно [14].

Вигляд інтерфейсу сервісу та тарифу наведені на рис. 3.6.

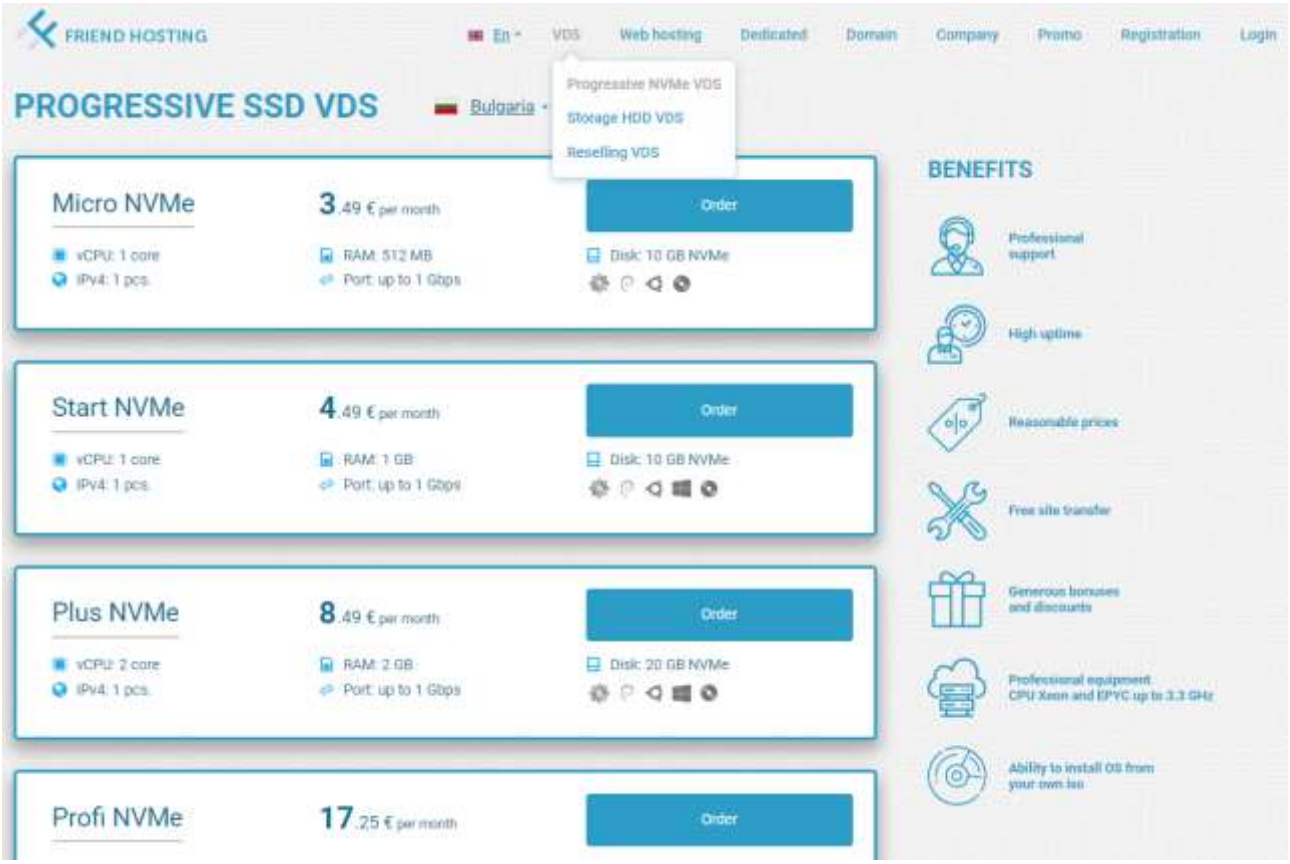

Рисунок 3.6 – Тарифні плани та інтерфейс сервісу Frient Hosting

При виборі параметрів серверу були встановлені параметри показані на рис. 3.7.

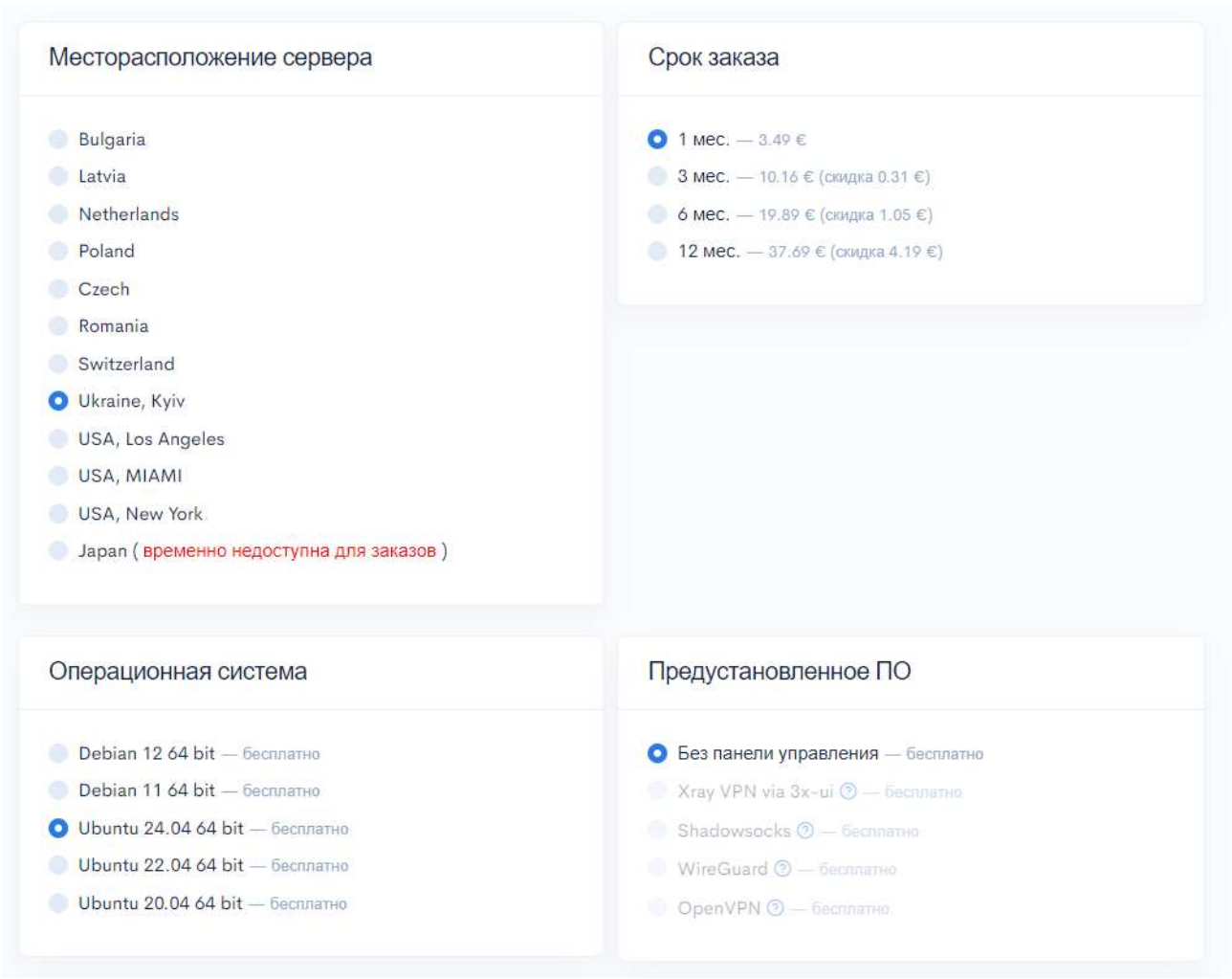

Рисунок 3.7 – Параметри налаштування ОС серверу

VDS (Virtual Dedicated Server), або віртуальний відділений сервер, відомий також як віртуальне відведене хостинг, пропонує більш відділене середовище порівняно з VPS. Він надає віртуальний сервер, який імітує досвід відведеного сервера, забезпечуючи постійну продуктивність. Це схоже на власний поверх в спільному будинку, з відведеними ресурсами [15].

Після оформлення замовлення було отримано дані для підключення до VDS серверу, які показано у тиблиці 3.1.

Таблиця 3.1

Дані для підключення до VDS

Назва параметру	Значення
IP-адреса	185.253.44.51
Логін	root
Пароль	
Порт	3333

Тепер маючи ці дані, можливо завантаження серверного додатку який буде розгортатися на сервері та до якого можна підключитися використовуючи IP-адресу 185.253.44.51 та порт на якому працює додаток. Наприклад, якщо запустити додаток на порту 8181 через VDS сервер, доступ до нього буде відкритий за посиланням http:// 185.253.44.51:8181/ або https:// 185.253.44.51:8181/ в залежності від налаштувань розгорненого додатку.

Для того щоб під'єднатися до серверу, налаштувати його та завантажити програмне забезпечення треба використовувати дані отримані після оформлення замовлення та підключитися до серверу використовуючи SSH (Secure Shell).

SSH (Secure Shell) - це криптографічний протокол для безпечного віддаленого з'єднання з комп'ютером або сервером через незахищену мережу, таку як Інтернет. Основна мета SSH - забезпечити захищене з'єднання, зашифроване і аутентифіковане, між клієнтом і сервером [16].

Основні особливості SSH включають:

- всі дані, які передаються між клієнтом і сервером, шифруються, що унеможливлює прослуховування або зміну даних третьою стороною;

- SSH використовує механізми аутентифікації, такі як паролі, ключі або сертифікати, щоб переконатися, що тільки правомірні користувачі мають доступ до системи;

- SSH дозволяє створювати безпечні тунелі для передачі інших протоколів, таких як FTP, HTTP або VNC, через захищене з'єднання;

- підтримка різних операційних систем: SSH підтримується на багатьох платформах, таких як Unix, Linux, Windows, MacOS і інші.

У випадку віддаленого підключення до сервера через SSH, ви можете виконувати команди на сервері, керувати файлами та налаштуваннями, а також виконувати інші адміністративні завдання, знаходячись на віддаленому комп'ютері або пристрої.

PuTTY - це безкоштовний і відкритий термінал для віддаленого управління комп'ютерами, який працює під операційними системами Windows і Unix. Він широко використовується для з'єднання з серверами за допомогою протоколів SSH, Telnet, Rlogin та RAW. PuTTY дозволяє адміністраторам віддалено керувати і конфігурувати сервери через командний рядок, надаючи доступ до файлів, виконання команд та роботу з системними налаштуваннями [17].

Після підключення необхідно завантажити програмне забезпечення і інструменти необхідні для роботи веб серверу:

Встановлення Java:

sudo apt install openjdk-11-jdk

Встановлення Gradle:

Gradle:

apt install zip

curl -s "https://get.sdkman.io" | bash

sdk install gradle 7.6.1

Після цього потрібно завантажити додаток. Для цього рекомендується використовувати систему контролю версій Git, яка дозволить легко завантажувати і оновлювати код додатку на віддаленому.

"Система контролю версій" - це система, що фіксує зміни в файлі або групі файлів з часом, дозволяючи вам отримати доступ до конкретних версій пізніше. Це особливо корисно в розробці програмного забезпечення, де файли вихідного коду зазвичай піддаються версійному контролю. Однак це можна застосовувати до різних типів файлів на комп'ютері.

"Git" - це система керування версіями, яка дозволяє вам відстежувати зміни в файлах та спільно працювати над ними з командою. Його основна ідея полягає в тому, що він зберігає повний набір даних про кожну версію проекту, а не тільки зміни між версіями. Це дозволяє вам легко повертатися до попередніх версій, знаходити помилки та вирішувати конфлікти у вашому коді [18].

Після завантаження додатку на віддалений сервер, достатньо перейти у проект, і ввести команду:

gradle run --no-daemon --max-workers=1

Для автоматизації процесу розгортання додатку на сервері слід використовувати systemctl - це інструмент управління службами у системі Linux, який використовується для керування ініціалізацією та управлінням процесами. Він дозволяє запускати, припиняти, перезапускати, відновлювати та перевіряти статус різних служб, таких як сервери, програми, фонові процеси тощо.

Команди systemctl дозволяють керувати системними службами, такими як служби ініціалізації (які запускаються під час завантаження системи), служби, що працюють в фоні (наприклад, веб-сервери або бази даних) та інші компоненти системи, які можуть бути управляються як служби [19].

Для того щоб автоматизувати запуск програми треба виконати наступні дії:

Створити файл з розширенням .service по шляху /etc/systemd/system/, наприклад remote-control-server.service і заповнити його наступним чином:

Description=My Kotlin Web Server

After=network.target

[Service]

ExecStart=/root/.sdkman/candidates/gradle/current/bin/gradle -p /root/web/remote-control-server/ run --no-daemon --max-workers=1

WorkingDirectory=/root/web/remote-control-server

User=root

Restart=always

RestartSec=10

[Install]

WantedBy=multi-user.target

Далі потрібно перезавантажити процеси утиліти командою: sudo systemctl daemon-reload.

Тепер для того щоб запустити сервер достатньо скористатися командою: sudo systemctl start remote-control-server.service.

Можлива помилка при виконанні команди зв’язана з тим що на найнижчиг тарифних планах хостингу на сервері використовується не велика кількість пам’яті, у такому випадку доцільно створити swop на Unix подібних операційних системах.

Swap (також відомий як swap space, обмінний простір або підкашування) - це область на твердому диску, яка використовується операційною системою для розширення обсягу доступної оперативної пам'яті (RAM). Коли весь фізичний обсяг пам'яті (RAM) використовується або близький до максимального рівня, операційна система може використовувати обмінний простір для зберігання неактивних частин пам'яті. Це дозволяє системі ефективно керувати ресурсами пам'яті і запобігає збою програм через нестачу пам'яті.

Щоб створити swop з розміром 1 гігабайт виконайте наступні команди:

sudo fallocate -l 1G /swapfile

sudo chmod 600 /swapfile

sudo mkswap /swapfile

sudo swapon /swapfile

Також у випадку недостатнього розміру оперативної пам’яті віддаленого серверу рекомендується додати наступні параметри у gradle.properties файл проекту:

org.gradle.daemon=false

org.gradle.parallel=false

org.gradle.configureondemand=false

org.gradle.jvmargs=-Xmx256m -Xms128m

3.3. Створення і опис архітектури мобільного додатку

Для створення мобільного додатку було обрано операційну систему Android і мову програмування Kotlin. Обираючи Android Kotlin серед інших різновидів технологій для розробки мобільних додатків, є декілька вагомих причин: Kotlin став дуже популярним в Android-спільноті, і його використання набуває широкого розповсюдження. Велика кількість розробників вибирають Kotlin через його сучасність, простоту використання і широкі можливості. Ця мова програмування також офіційно підтримується Google для розробки Android додатків, що робить його перевагу для тих, хто працює у середовищі Android. Також він пропонує багато корисних функцій, таких як розширення функцій, лямбда-вирази, корутини (coroutines) для асинхронного програмування та інші. Ці можливості дозволяють писати більш зрозумілий, ефективний та легко збережений код.

Бібліотеки для реалізації ідентифікації та автентифікації через веб-сервер наведені у таблиці 3.3.

Таблиця 3.3

Android Kotlin бібліотеки для реалізації завдання роботи

Бібліотека	Значення
Retrofit	Бібліотека для виконання HTTPS-запитів у Android-додатках. Надає зручний інтерфейс для роботи з веб-сервісами, включаючи зручне парсінг та обробку відповідей. Використовується для запитів на веб-сервер
OkHttp	Використовується для виконання мережевих запитів у Android-додатках. Але у даному додатку вона відповідає за підключення і конфігурування бібліотеки Retrofit, для коректних запитів із використовуванням SSL сертифікату

Hilt Dagger	Бібліотека для реалізації залежностей у Android-додатках за допомогою паттерну внедрення залежностей (DI). Вона спрощує створення та управління залежностями у додатку.
Navigation	Бібліотека використовується для навігації між екранами (фрагментами) у Android-додатках. Вона надає зручний спосіб керувати переходами між екранами та передачею даних між ними.
Glide	Бібліотека для завантаження та відображення зображень у Android-додатках. Вона дозволяє зручно та ефективно працювати з зображеннями, включаючи завантаження з Інтернету та кешування.

Сучасна розробка Android додатків орієнтується на розробку з використанням архітектури додатку Clean Architecture.

Clean Architecture - це підхід до розробки програмного забезпечення, запропонований Робертом Мартіном. Вона пропонує структуру, яка сприяє розробці гнучкого, тестованого та підтримуваного коду. Clean Architecture фокусується на забезпеченні незалежності компонентів від зовнішніх сутностей, таких як фреймворки, бібліотеки чи інтерфейси. Основна ідея полягає в тому, щоб помістити логіку бізнес-правил у центр архітектури, ізолювати її від зовнішніх впливів і залежностей. Це дозволяє легко модифікувати зовнішні компоненти, такі як інтерфейси користувача або джерела даних, не торкаючись основної бізнес-логіки. Використання Clean Architecture має кілька переваг, включаючи кращу тестованість, легкість підтримки та розширення, а також можливість повторного використання компонентів у різних проектах. Вона забезпечує чітке розділення відповідальностей між шарами архітектури, що сприяє більшій гнучкості та модульності. Clean Architecture особливо корисна для великих проектів, оскільки вона допомагає зберігати код організованим і керованим у міру зростання додатку. Загалом, використання Clean Architecture

дозволяє розробникам створювати стійкий, тестований і масштабований код, що полегшує його підтримку та розширення в майбутньому [20].

Діаграма варіантів використання наведена на рис. 3.8. Вона слугує орієнтиром на який опирається побудова архітектури додатку.

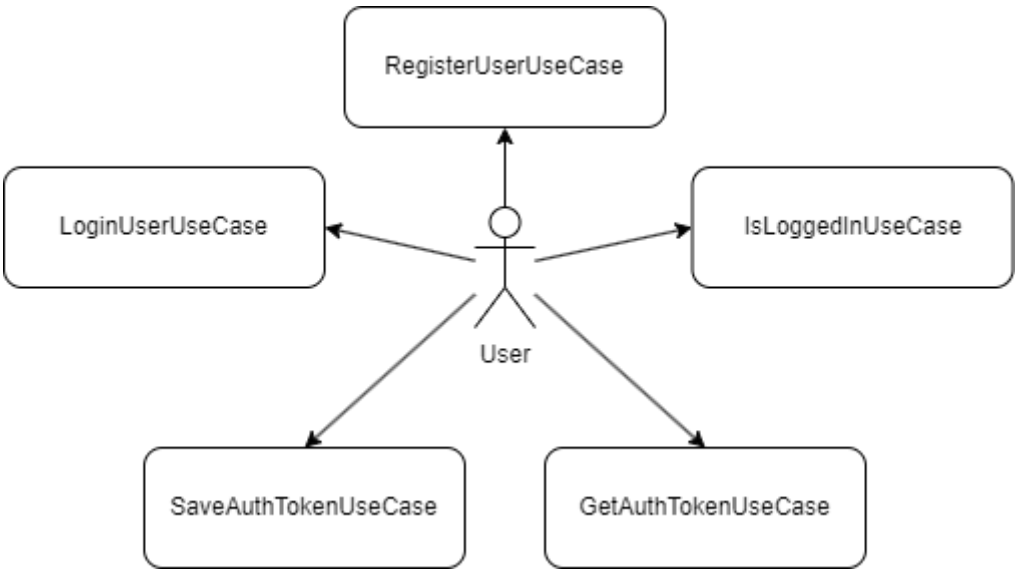

Рисунок 3.8 – Діаграма варіантів використання

Беручи до уваги усе розглянуте вище: веб-сервер, налаштування зовнішнього серверу, постанову задачі, архітектуру і діаграми, створено архітектуру відповідно моделі Clean Architecture.

Загальну схему концепції модулів програми з Clean Architecture зазначено на рис. 3.9.

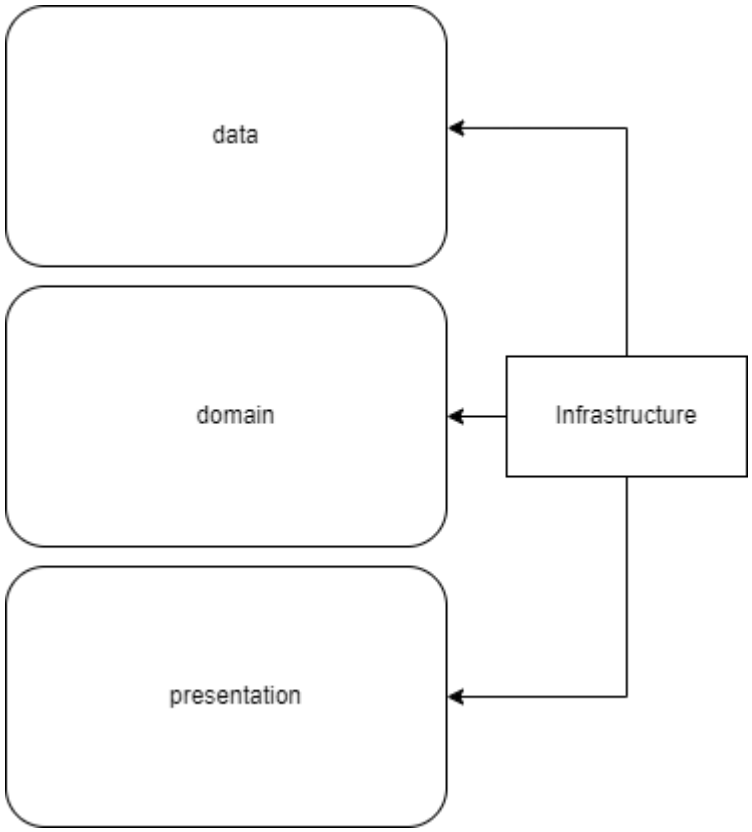

Рисунок 3.9 – Загальна схема концепції модулів програми з Clean Architecture

У Clean Architecture існує чіткий поділ на шари або компоненти, кожен з яких має свою відповідальність та спосіб взаємодії з іншими компонентами.

Domain:

Цей компонент знаходиться в центрі архітектури та містить бізнес-логіку програми. Тут визначаються сутності (моделі даних), інтерфейси репозиторіїв та використовувані бізнес-правила. Domain не залежить від інших компонентів і не містить жодних зовнішніх залежностей. Це ядро програми, яку можна використовувати у будь-якому контексті.

Data:

Компонент Data відповідає за отримання та обробку даних із різних джерел, таких як бази даних, API або файлові системи. Він реалізує інтерфейси репозиторіїв, визначені в Domain, та відповідає за перетворення даних у моделі, зрозумілі для Domain. Data може містити класи, які взаємодіють із зовнішніми залежностями, такими як бібліотеки для роботи з базами даних або запитами мережі.

Presentation:

Цей компонент є користувальницьким інтерфейсом програми, такою як активіти, фрагменти або ViewModels в Android. Він взаємодіє з компонентом Domain через інтерфейси та використовувані в ньому бізнес-правила для відображення даних користувача. Presentation також може містити логіку форматування даних для відображення та обробки подій користувача.

Infrastructure:

Компонент Infrastructure містить допоміжні класи та утиліти, які можуть використовуватись іншими компонентами. Наприклад, тут можуть бути класи для роботи з мережею, базами даних, бібліотеки для відображення зображень та інші спільні бібліотеки або фреймворки, що використовуються в програмі.

Зв'язок між компонентами здійснюється наступним чином:

Domain не залежить від інших компонентів та визначає інтерфейси репозиторіїв.

Data реалізує інтерфейси репозиторіїв, визначені в Domain, та надає дані із зовнішніх джерел.

Presentation використовує бізнес-логіку з Domain і отримує дані з Data для відображення інтерфейсу користувача.

Infrastructure надає допоміжні класи та утиліти для використання іншими компонентами.

Такий поділ на компоненти дозволяє створювати модульний, тестований код, що легко підтримується. Зміни в одному компоненті не впливають на інші, що полегшує внесення змін та додавання нової функціональності до програми.

Clean Architecture це концепції по яким вибудовується програмне забезпечення і відповідно до яких повинна відповідати архітектура додатку.

Для побудови архітектури слід використовувати шаблон MVVM (Model-View-ViewModel) - це архітектурний шаблон, що широко використовується в розробці додатків, особливо на платформі Android. Він поділяє додаток на три основні компоненти: Model, View та ViewModel. Розглянемо кожен із них докладніше:

Model:

Модель є шаром даних вашої програми. Вона містить класи, що описують структуру даних, бізнес-логіку та правила валідації. Модель не має залежності від інших компонентів MVVM і часто взаємодіє з репозиторіями даних або API.

View:

View відповідає за відображення інтерфейсу користувача та взаємодію з ним. В Android це можуть бути Activity, Fragment або View користувача. Подання не містить жодної бізнес-логіки і служить виключно для відображення даних і обробки подій (натискань, введення тексту і т.д.).

ViewModel:

ViewModel є сполучною ланкою між Моделью та Поданням. Він отримує дані з Моделі, перетворює їх у потрібний формат і надає Подання. ViewModel також обробляє події введення користувача з Подання та виконує відповідні дії, такі як виклик бізнес-логіки в Моделі або оновлення даних у Поданні.

Взаємодія між компонентами відбувається так:

View пов'язані з ViewModel через механізм прив'язки даних Data Binding чи спостерігачів Observers.

ViewModel отримує дані з Model і перетворює їх у формат, зручний для відображення у Поданні.

Коли користувач взаємодіє з View, воно сповіщає ViewModel про події (натискання кнопок, введення тексту і т.д.).

ViewModel обробляє ці події, викликаючи відповідні методи в Моделі або оновлюючи дані у Поданні.

Модель (Model) виконує бізнес-логіку та правила валідації, а потім повертає оновлені дані назад у ViewModel.

ViewModel передає оновлені дані до Подання для відображення.

Такий підхід дозволяє чітко розділити відповідальності між компонентами та забезпечує просту тестованість кожного з них. Подання відповідає тільки за інтерфейс користувача, ViewModel за логіку подання, а Модель за бізнес-логіку та правила валідації даних.

На рисунку 3.10 відтворена загальна архітектура додатку відповідно MVVM.

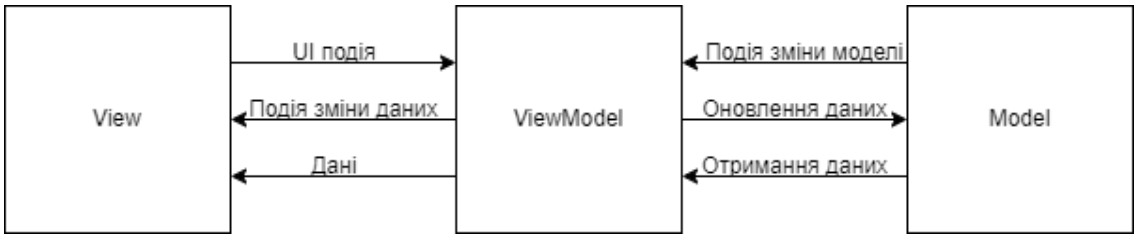

Рисунок 3.10 – Загальна архітектура додатку відповідно MVVM

Якщо об’єднати MVVM і Clean Architecture і проектуванні з переходом на більш низькі рівні абстракції, отримаємо чисту архітектуру, наведену на рис. 3.11.

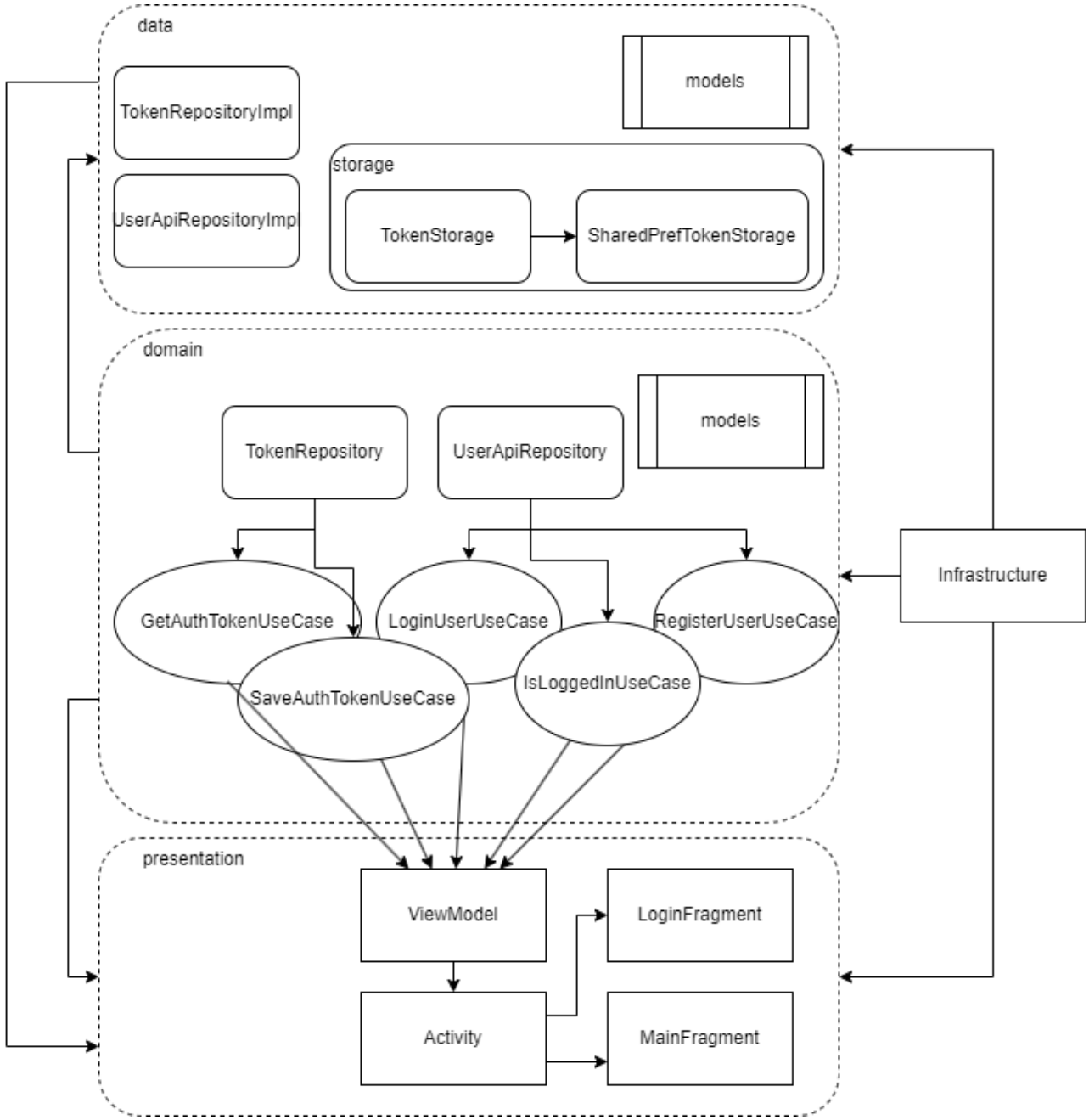

Рисунок 3.11 – Архітектура Android додатку

Для авторизації через веб-сервер, Android додаток зберігає лише JWT токен, який надається йому після проходження ідентифікації і авторизацій. Пошта і пароль ні в якому разі не зберігається локально на пристрої, зберігається лише тимчасовий JWT токен доступу, який перевіряється при відпритті додатку.

Для зберігання JWT використовується SharedPreferences - це механізм збереження приватних даних в Android додатках у вигляді пар ключ-значення. Цей механізм дозволяє зберігати налаштування, стан додатка та інші дані, які

потрібні для подальшого використання. SharedPreferences мають механізми захисту, які дозволяють зберігати дані у вигляді шифрованого тексту. Це важливо для зберігання конфіденційної інформації, такої як токени доступу.

Для реалізації перевірки при відкритті додатку, і під час розгорнення додатку з недавніх додатків, використана функціональність життєвого циклу компоненту Activity.

Перевизначення методу onResume(…) дає змогу відловлювати подію відкриття і розгортання додатку.

Для асинхронних операцій було прийнято рішення використовувати sealed class синтаксису Kotlin, який схожий на сигнатуру enum відомих мов програмування, але дає можливість вичерпно виділити тип сутності та ідентифікувати її.

Код вводить sealed class AsyncResult, який дозволяє представляти результат асинхронної операції одним із трьох станів: PendingResult (операція в процесі), SuccessResult (операція завершена успішно) або ErrorResult (операція завершена з помилкою). Такий підхід допомагає структурувати обробку асинхронних операцій та покращує читаність коду.

Використання типізованих функціональних типів:

Визначення typealias Mapper<Input, Output> = (Input) -> Output вводить псевдонім для типу функції, що приймає один аргумент типу Input і повертає значення типу Output. Це дозволяє більш компактно та чітко виражати перетворення даних за допомогою функцій вищого порядку.

Функція map для перетворення результатів:

Функція map визначена як розширення AsyncResult і дозволяє застосовувати перетворення (Mapper) до результату асинхронної операції. Це дає можливість перетворювати дані, що повертаються успішною операцією, перед їх подальшим використанням.

Безпечна обробка null:

У реалізації SuccessResult використовується безпечний виклик ?.let { ... } для обробки випадку, коли data є null. Це запобігає виникненню

NullPointerException при спробі застосувати перетворення (mapper) до значення null.

Функція takeData для безпечного вилучення даних:

Функція takeData дозволяє безпечно отримати дані з SuccessResult. Якщо поточний об'єкт не є SuccessResult, функція поверне null, що допомагає уникнути помилок під час роботи з результатом операції.

Використання такого коду забезпечує більш структуровану та безпечну обробку асинхронних операцій, дозволяючи чітко розділяти різні стани результату та застосовувати необхідні перетворення даних за допомогою функцій вищого порядку. Це покращує читаність, підтримуваність та безпеку коду, що працює з асинхронними операціями.

Відправка медіа файлів і комунікація між Android та Windows додатками зі сторони мобільного ПЗ відбувається наступним чином:

Завантаження списку фотографій з галереї пристрою:

- у класі MainFragment є функція getAllPhotos(), яка використовує MediaStore.Images.Media.EXTERNAL_CONTENT_URI для отримання шляхів до фотографій з галереї пристрою;

- ця функція повертає список об'єктів Photo, які містять шлях до файлу зображення;

- список фотографій завантажується в RecyclerView через адаптер PhotoAdapter.

Передавання вибраної фотографії на сервер:

- при натисканні на кнопку "Send" у фрагменті PhotoDetailsFragment викликається функція FileTransferTask().execute() з параметрами IP-адреси сервера, порту сервера та шляху до файлу фотографії;

- всередині FileTransferTask відбувається з'єднання з сервером через сокет за вказаною IP-адресою та портом;

- дані файлу (ім'я файлу, розмір файлу та вміст файлу) передаються на сервер через вихідний потік сокета.

3.4. Створення і опис архітектури додатку для комп’ютера

Для створення додатку для комп’ютера який буде приймати медіа файли було обрано мову програмування C# і технологію Win Forms. Такий вибір зумовлений наступними перевагами:

- .NET Framework, на якій базується C#, є кросплатформенним середовищем виконання, що дозволяє запускати додатки на різних операційних системах, включаючи Windows, Linux та macOS (за допомогою Mono). Хоча Windows Forms орієнтована на розробку додатків для Windows, їх також можна запускати на інших платформах за допомогою відповідних інструментів;

- C# є сучасною об'єктно-орієнтованою мовою програмування, що забезпечує багатий набір функцій, таких як спадкування, інкапсуляція, поліморфізм та багато іншого. Це дозволяє створювати модульний, легко підтримуваний та розширюваний код.

- також .NET Framework надає багато вбудованих бібліотек та API для вирішення найрізноманітніших завдань, таких як робота з мережею, файлами, базами даних, графікою тощо. Це спрощує розробку та зменшує написання стандартного низькорівневого коду;

- Windows Forms є частиною .NET Framework і надає готову інфраструктуру для створення графічних користувацьких інтерфейсів для додатків на Windows. Це дозволяє легко розробляти настільні додатки з багатим функціоналом та зручним інтерфейсом;

- .NET Framework включає вбудовані механізми безпеки, такі як керований код, автоматичне керування пам'яттю (Garbage Collector) та захист від небезпечного коду. Це зменшує ризики безпеки та полегшує розробку надійних додатків;

- що найголовніше, оптимізовість для високої продуктивності та можливість виконувати Just-In-Time (JIT) компіляцію коду для підвищення швидкодії.

Для передачі даних у додатку реалізований запуск локального TCP серверу, який працює у на комп’ютері і до якого можуть під'єднуватися пристрої які знаходяться у спільній локальній мережі.

TCP сокети є досить простим і прямим способом передачі даних між клієнтом і сервером. Це базовий рівень мережевих протоколів, який дозволяє передавати будь-які типи даних, включаючи файли. Але, в свою чергну, TCP забезпечує надійну передачу даних, гарантуючи доставку пакетів та їх правильну послідовність. Це важливо для передачі файлів, щоб жодні дані не втрачалися та не перемішувалися. Також, наявність вбудований механізм контролю потоку, який запобігає перевантаженню сервера або клієнта даними дозволяє передавати великі файли, не ризикуючи втратити дані або перевантажити систему.

3.4. Висновки до третього розділу

Було розглянуто опис програмного забезпечення, методи і розробка архітектури веб-серверу. Розглянутий та аргументований вібір мов програмування для додатків. Відтворені моделі та діаграми архітектури для побудови якісного та безпечного програмного забезпечення.

Виокремлюючи логічні розділи можна визначити наступні висновки:

Створення веб-серверу:

- розглянуті прийняті рішення щодо запровадження безпечної ідентифікації та автентифікації користувачів;
- дослідження інструментів і бібліотек запровадження безпечної передачі даних;
- розглянута реалізація веб-серверу із застосуванням криптоалгоритмів;
- аналіз сучасних підходів до розробки програмного забезпечення веб-серверу та їх вибір для реалізації.

Створення віддаленого серверу:

- розглянуті ресурси і хостинг постачальники, проведений аналіз та обраний найоптимальніший;

- аргументація вибору хостинг провайдеру, тарифу та параметрів сервера

- зазначено методи встановлення, завантаження та розгортання додатку на сервері;

- розглянуто можливі помилки і складнощі під час роботи з віддаленим сервером та їх вирішення.

Створення мобільного додатку:

- розглянуті сучасті методи та принципи побудови архітектури;

- розглянуті використані для реалізації бібліотеки та їх роль у реалізації завдання;

- виявлені методи збереження локально та відпралення даних через HTTPS, Socket.

Створення додатку для комп’ютера:

- розглянуто оптимальність вибору методу створення додатку;

- аргументовано вибір підходу до утворення локального серверу;

- перелічено переваги використання TCP для обміну медіа даними між мобільним додатком та додатком комп’ютера.

ВИСНОВОК

Мета роботи, що складається з створення системи забезпечення віддаленої комунікації з використанням криптоалгоритмів захищених каналів досягнута.

Проведено дослідження сучасних методів захищення каналів управління, та обрано оптимальні методи реалізації програмного забезпечення ідетифікації та автентифікації для додатку передачі медіа даних, що дозволяє ефективно забезпечити безпеку та конфіденційність користувацької інформації.

Застосування криптографічних методів та авторизації доступу за ролями користувачів значно знижує ризики несанкціонованого доступу та витоку даних. Водночас реалізована система надає можливість зручного відправлення медіа даних завдяки використанню локальних мереж, що може підвищити зручність комунікації мобільного телефону та комп'ютеру.

Кваліфікаційна робота виконана у відповідності до стандарту спеціальності 121 – «Інженерія програмного забезпечення» і демонструє володіння такими компетентностями як:

- здатність розробки специфічних моделей програмного забезпечення на основі існуючих парадигм програмування, підходів до розробки інформаційних систем, а також безпеки і зручності кінцевого користувача;

- здатність самостійно вивчати і аналізувати варіанти реалізації системи, а також впроваджувати і використовувати передові наукові відкриття у розробці;

- здатність реалізувати багаторівневу модель комунікації між компонентами системи, побудувати та налагодити їх цілісність та спільну комунікацію.

Серед результатів навчання, визначених стандартом кваліфікаційна робота реалізовує наступні:

- будувати архітектурної моделі програмного забезпечення відповідно раціональності часових та ресурсних витрат, з передбаченням подальшого розвитку системи;

- застосовувати концепцій побудови та моделювання баз даних

- використовувати сервіси та інструментальні засоби для побудови мережевих серверних частин, їх захисту та зв’язку з іншим програмним забезпеченням;

- використовувати та будувати алгоритми криптографічних алгоритмів захисту інформації;

- використовувати парадигми програмування для забезпечення оптимального побудування інформаційних систем.

Розроблене програмне рішення відповідає вимогам безпеки в галузях, де працюють з конфіденційною інформацією, завдяки використанню надійних криптоалгоритмів таких як RAS, AES і використанню токену JWT. Впровадження даної системи дозволяє спростити та водночас посилити захист процесів передачі та обробки даних, необхідних для ідентифікації користувачів. Робота клалася з трьох компонентів: веб-серверу, мобільного додатку та додатку на комп’ютер. Веб’сервер був розроблений за допомогою мови програмування Kotlin і фреймворків Spring, Spring Security за допомогою яких було реалізовано ідентифікацію та автентифікацію користувачів використовуючи реляційну базу даних MySql комунікація з якою відбувалась через бібліотеку JPA. Також у веб-сервері наведена програмна реалізація підключення згенерованого SSL сертифікату і використання JWT для забезпечення безпеки. На мобільному додатку у свою чергу була розроблена клієнтська частина яка відповідала за надсилання запитів на веб-сервер, зберігання JWT токену, використання локального серверу для відправлення медіа файлів на комп’ютер використовуючи TCP сервер реалізований на C#.

Отримані результати мають практичне значення, оскільки методи та алгоритми ідентифікації та автентифікації можуть бути впроваджені у будь які додатки, необхідним у роботі з даними користувача. Також дана робота може слугувати прикладом і роз’ясненням того як реалізовується використання криптоалгоритмів і захищені канали управління, а також підключення CT та створення додатку з JWT токеном.

З можливостей покращення можна відмітити впровадження більш гнучкого і зручного інтерфейсу по відправленню медіа даних, а також додавання refresh JWT токену для забезпечення додаткового захисту і контролю даних користувачів.

Отже, створена система є актуальним та практично корисним рішенням, що може забезпечити належний рівень безпеки при роботі з персональними даними в різноманітних додатках відповідно до сучасних вимог.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Pravir C. Network Security with OpenSSL / C. Pravir, M. Matt, V. John., 2002. – (0-596-00270-X).

2. SSL/TLS and PKI History [Електронний ресурс]. – 2023. – Режим доступу до ресурсу: https://www.feistyduck.com/ssl-tls-and-pki-history/.

3. Mehwish A. Securing Web Transactions TLS Server Certificate Management / Akram Mehwish. – U.S. Department of Commerce, 2020. – 432 с. – (NIST SPECIAL PUBLICATION 1800-16).

4. Joshua D. Implementing SSL/TLS Using Cryptography and PKI / Davies Joshua. – Crosspoint Boulevard Indianapolis: Wiley Publishing, Inc., 2011. – 697 с.

5. Let's Encrypt Continues Partnership with Princeton to Bolster Internet Security [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://letsencrypt.org/2024/05/30/princeton-partnership.

6. Capture HTTPS traffic using the Postman built-in proxy [Електронний ресурс] – Режим доступу до ресурсу: https://learning.postman.com/docs/sending-requests/capturing-request-data/capturing-https-traffic/.

7. SOAP [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/TR/soap/.

8. GraphQL [Електронний ресурс] // GraphQL contributors. – 2021. – Режим доступу до ресурсу: https://spec.graphql.org/October2021/.

9. ANDREW D. B. Implementing Remote Procedure Calls / D. B. ANDREW, J. N. BRUCE.

10. The WebSocket Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://datatracker.ietf.org/doc/html/rfc6455.

11. Representational State Transfer (REST) [Електронний ресурс] – Режим доступу до ресурсу: https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

12. Security Keys [Електронний ресурс] // ImperialViolet. – 2018. – Режим доступу до ресурсу: https://www.imperialviolet.org/2018/03/27/webauthn.html.

13. Enable HTTPS on your servers [Електронний ресурс] – Режим доступу до ресурсу: https://web.dev/articles/enable-https?hl=en&visit_id=638526643627612706-3970929105&rd=1.

14. FRIEND HOSTING [Електронний ресурс] – Режим доступу до ресурсу: https://friendhosting.net/en.

15. VPS vs VDS vs Dedicated Servers: Choosing the Right Hosting Solution [Електронний ресурс] // medium. – 2023. – Режим доступу до ресурсу: https://medium.com/@stephenhodgkiss1/vps-vs-vds-vs-dedicated-servers-choosing-the-right-hosting-solution-836901014185.

16. Daniel J. Barrett. SSH, The Secure Shell: The Definitive Guide / Daniel J. Barrett, Richard E. Silverman, Robert G. Byrnes., 2005. – (Second Editing).

17. Download PuTTY [Електронний ресурс] – Режим доступу до ресурсу: https://www.putty.org/.

18. Scott C. Pro Git / C. Scott, S. Ben., 2024. – 501 с. – (2.1.429).

19. Linux: systemctl – управління службами [Електронний ресурс] // rtfm. – 2016. – Режим доступу до ресурсу: https://rtfm.co.ua/ru/linux-systemctl-upravlenie-sluzhbami/.

20. Robert C. M. Clean Architecture: A Craftsman’s Guide to Software Structure and Design / C. Martin Robert., 2017. – 352 с. – (1st Edition). – (978-0134494166).