НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
Інформаційних систем і технологій
_____ Швиденко М. З., к. ек. н., доцент
(підпис) (ПІБ, вчене звання і ступінь)

«__»_____2025р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка корпоративного месенджера для аграрних підприємств»

Спеціальність 122 «Комп'ютерні науки»

Гарант освітньої програми
д. ек. н, професор _____ Руденський Р.А.
(науковий ступень та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
д. пед. н., професор _____ Кузьмінська О.Г.
(науковий ступень та вчене звання) (підпис) (ПІБ)

Виконав _____ Самарін І.В.
(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інформаційних систем і технологій

_____ Швиденко М. З., к. ек. н., доцент

(підпис) (ПІБ, вчене звання і ступінь)

«__»_____2025р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи студенту

САМАРІНУ ІЛЛІ ВАЛЕРІЙОВИЧУ

1. Тема проєкту «Розробка корпоративного месенджера для аграрних підприємств», затверджена наказом ректора НУБіП України від 24.04.2025р. №684”С”
2. Термін подання завершеної роботи на кафедру 2025.06.02
   (рік, місяць, число)
3. Вихідні дані до проєкту: база даних MySQL, серверна частина, програмний та веб додатки, сформований опис проєкту.
4. Перелік питань що розглядаються:
   - Аналіз предметної області та постановка завдання;
   - Проєктування корпоративного месенджера;
   - Розробка корпоративного месенджера;
   - Тестування корпоративного месенджера;
   - Впровадження і експлуатація системи;
   - Висновки.
5. Перелік графічного матеріалу:
   - Діаграма послідовностей;
   - Діаграма прецедентів;
   - ER діаграми бази даних;
   - Діаграма розгортання;
   - Діаграма розгортання.

Дата видачі завдання “ 24 ” квітня 2025р.

Керівник бакалаврської кваліфікаційної роботи

Доктор пед. н., професор _____ Кузьмінська О.Г.

Виконав _____ Самарін І.В.

АНОТАЦІЯ

У дипломному проєкті розглянуто розробку корпоративного месенджера для аграрних підприємств, адаптованого до умов територіальної розподіленості, нестабільного інтернету та багаторівневої організаційної структури. Система забезпечує захищений обмін повідомленнями, підтримує групові чати, кросплатформені клієнти та офлайн-доступ. У роботі описано вимоги до системи, технічне завдання, архітектуру, реалізацію серверної та клієнтської частини, базу даних, а також особливості впровадження в аграрному секторі.

ANNOTATION

The diploma project focuses on developing a corporate messenger tailored for agricultural enterprises, considering their distributed structure, unstable internet connectivity, and hierarchical organization. The system provides secure messaging, supports group chats, cross-platform clients, and offline access. The work presents system requirements, technical specifications, architecture, implementation of both server and client components, database design, and deployment specifics in the agricultural sector.

Зміст

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

- HTTP – HyperText Transfer Protocol
- HTTPS – HyperText Transfer Protocol Secure
- SQL – Structured Query Language
- CSS – Cascading Style Sheets
- HTML – Hypertext Markup Language
- SSL – Secure Sockets Layer
- TLS – Transport Layer Security
- API – Application Programming Interface
- REST – Representational State Transfer
- JSON – JavaScript Object Notation
- QML – Qt Meta Language або Qt Modeling Language
- URL – Uniform Resource Locator

ВСТУП

У сучасному світі месенджери стали невід’ємною частиною цифрової комунікації — як у повсякденному житті, так і в професійній діяльності. Їх використання дозволяє значно підвищити ефективність обміну інформацією, забезпечити швидку координацію дій і спростити взаємодію між учасниками різних процесів. Особливо важливу роль месенджери відіграють у сфері корпоративного управління, де своєчасна комунікація між співробітниками напряму впливає на продуктивність роботи підприємства.

Аграрний сектор, як одна з ключових галузей економіки України, все частіше впроваджує сучасні інформаційні технології для підвищення ефективності управління виробничими процесами. У великих сільськогосподарських підприємствах задіяна значна кількість працівників, які працюють у різних географічних локаціях: в офісі, на складах, у полі, на фермах. Це створює потребу в надійному, швидкому та безпечному засобі внутрішньої комунікації, що дозволяє координувати дії в реальному часі.

Корпоративний месенджер є ефективним рішенням для таких задач. Він забезпечує обмін текстовими, голосовими та мультимедійними повідомленнями, підтримує групові чати, дозволяє створювати канали для внутрішніх оголошень, а також інтегрується з системами планування та обліку. Завдяки мобільності й кросплатформенності, месенджер може бути доступним на будь-якому пристрої з підключенням до Інтернету, що особливо важливо для працівників, які виконують свої обов’язки в польових умовах.

Окрему увагу при розробці корпоративного месенджера приділено питанням безпеки даних та конфіденційності. Використання сучасних протоколів шифрування, автентифікації та контролю доступу дозволяє захистити інформацію підприємства від несанкціонованого доступу.

Таким чином, створення корпоративного месенджера для аграрних підприємств є актуальним завданням, що відповідає потребам галузі в сучасних умовах.

11 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

11.1 Актуальність теми

У XXI столітті цифрові технології активно впроваджуються в усі галузі економіки, включаючи аграрний сектор. Згідно з аналітичними даними Світового банку та дослідженнями міжнародних організацій, впровадження інформаційних систем у сільське господарство сприяє зростанню продуктивності на 20–30% завдяки покращенню управлінських процесів, координації дій працівників та оперативному прийняттю рішень [1] . Одним із ключових напрямів цифровізації агросфери є створення ефективних засобів комунікації, зокрема внутрішніх корпоративних месенджерів.

Більшість аграрних підприємств мають розгалужену структуру, де працівники розміщені у віддалених локаціях: у полях, на фермах, у логістичних центрах та адміністративних офісах. Традиційні засоби зв’язку (мобільний телефон, електронна пошта) не забезпечують достатнього рівня швидкості, зручності та безпеки. Водночас популярні месенджери, такі як Telegram, Viber, WhatsApp чи навіть Slack, не враховують специфічних потреб аграрного бізнесу: нестабільне інтернет-з'єднання в сільській місцевості, потребу в офлайн-доступі до повідомлень, обмеження по енергоспоживанню та необхідність чіткого поділу за ролями й підрозділами.

За результатами дослідження компанії McKinsey, інтеграція внутрішніх комунікаційних систем у корпоративне середовище дозволяє знизити витрати часу на координацію до 30%, а рівень внутрішньої ефективності — підвищити на понад 25% [2]. Саме тому створення корпоративного месенджера, спеціально адаптованого під аграрні підприємства, є актуальним і стратегічно важливим завданням для галузі.

Запропонований проєкт «Розробка корпоративного месенджера для аграрних підприємств» має на меті створення програмного продукту, який

дозволяє організувати безпечну, швидку та адаптивну систему обміну інформацією для агропідприємств. Такий месенджер повинен враховувати не лише типові комунікаційні функції (повідомлення, файли, дзвінки), але й підтримку офлайн-режиму, розподіл за структурними підрозділами, а також високий рівень захисту даних, зокрема завдяки використанню шифрування, авторизації та локального кешування.

Отже, розробка спеціалізованого корпоративного месенджера для аграрного сектору відповідає глобальним тенденціям цифрової трансформації, сприяє підвищенню ефективності агровиробництва та є вкрай актуальною в умовах сучасного технологічного розвитку України.

11.2 Мета і завдання дослідження

Метою даного дослідження є створення програмного засобу — корпоративного месенджера, який забезпечуватиме ефективну, безпечну та адаптивну комунікацію між працівниками аграрного підприємства. Особливу увагу приділено умовам територіальної розподіленості персоналу, багаторівневій організаційній структурі підприємств та нестабільному інтернет-з’єднанню, що характерне для багатьох сільськогосподарських регіонів.

Для реалізації поставленої мети передбачається вирішення ряду завдань. На початковому етапі дослідження необхідно проаналізувати сучасний стан засобів комунікації, які вже використовуються в аграрному секторі, та визначити їх придатність для внутрішньої корпоративної взаємодії. Далі слід дослідити функціональні можливості наявних месенджерів, зіставити їх сильні й слабкі сторони з вимогами корпоративного використання.

На основі зібраної інформації формулюються технічні та функціональні вимоги до корпоративного месенджера, адаптованого до специфіки аграрної сфери. Після цього необхідно спроєктувати архітектуру майбутньої системи з урахуванням мультиплатформенності, високої продуктивності та вимог до інформаційної безпеки.

Наступним кроком є реалізація серверної частини, яка повинна підтримувати обробку запитів у реальному часі, забезпечувати безпечне зберігання даних, здійснювати автентифікацію користувачів і взаємодіяти з базою даних. Паралельно розробляється клієнтська частина для десктопних та мобільних платформ із зручним і гнучким інтерфейсом.

Після створення функціональних компонентів необхідно провести тестування системи в умовах, що максимально наближені до реального середовища роботи аграрного підприємства. Завершальним етапом є оцінка ефективності впровадженого рішення та визначення можливостей його подальшого розвитку з урахуванням технологічних змін та потреб галузі.

11.3 Обґрунтування мети та визначення завдань дипломного проєктування

Дипломне проєктування орієнтоване на створення прикладного програмного забезпечення, що має відповідати реальним потребам цільової аудиторії та вирішувати конкретні прикладні завдання. У межах цього проєкту розробляється корпоративний месенджер, призначений для аграрних підприємств з урахуванням їхніх організаційних особливостей, географічної розподіленості та обмежених технічних ресурсів. Основною метою є створення повноцінної клієнт-серверної системи, яка забезпечує зручну, безпечну та ефективну комунікацію між працівниками.

Месенджер повинен підтримувати передачу повідомлень у реальному часі, забезпечувати керування контактами й групами, надавати веб-доступ, а також відповідати сучасним вимогам безпеки. Для реалізації цієї мети визначено низку технічних та організаційних завдань.

Передусім необхідно розробити базу даних серверної частини, яка зберігатиме інформацію про користувачів, повідомлення, групи, сесії та інші сутності. Вона має бути побудована на основі СКБД MySQL з оптимізованою структурою та налагодженою взаємодією із сервером.

Наступним кроком є реалізація серверної частини на мові C++ із використанням бібліотек Boost.Beast та Asio, що дозволить обробляти запити асинхронно та підтримувати високу продуктивність. Сервер повинен надавати функціонал у вигляді REST API для взаємодії з клієнтами.

Особливу увагу приділено розробці клієнтської частини: вона має бути кросплатформенною, реалізованою за допомогою Qt Framework та QML [3], з інтуїтивно зрозумілим інтерфейсом, підтримкою обміну повідомленнями, перегляду історії чатів та управління контактами.

Окремим модулем розробляється веб-інтерфейс системи, створений із застосуванням HTML, CSS та JavaScript, що забезпечить можливість базової взаємодії з системою через браузер, включаючи реєстрацію, авторизацію та перегляд повідомлень.

Також передбачається використання сучасних інструментів розробки, таких як Visual Studio Code [4], та MySQL Workbench , для зручної організації процесу створення, тестування й підтримки програмного забезпечення.

У результаті реалізації проєкту буде створено комплексну систему корпоративної комунікації, яка охоплює сервер, клієнтські застосунки, базу даних і вебінтерфейс. Такий підхід дозволить аграрним підприємствам ефективно організовувати внутрішню взаємодію, зменшити втрати часу на передачу інформації та підвищити рівень цифрової інтеграції у виробничі процеси.

11.4 Предметна область та її моделювання

Об’єктом дослідження є процеси внутрішньої комунікації та інформаційного обміну в аграрних підприємствах із територіально розподіленою структурою, де комунікаційні потоки охоплюють як адміністративний персонал, так і працівників, задіяних у польових роботах.

Предметом дослідження виступають методи, засоби та технології розробки корпоративного програмного забезпечення для обміну повідомленнями. Зокрема, розглядаються архітектура системи, її функціональні компоненти, механізми безпеки та особливості адаптації до аграрної галузі.

У межах дипломного проєкту дослідження охоплює проєктування клієнт-серверної архітектури корпоративного месенджера, розробку механізмів автентифікації, шифрування та захищеної передачі даних. Особливу увагу приділено адаптації інтерфейсу користувача до умов сільськогосподарського виробництва, яке передбачає роботу в польових умовах, часто з використанням мобільних пристроїв з обмеженими ресурсами.

Моделювання предметної області включає аналіз структури підприємства, типових сценаріїв взаємодії між працівниками, необхідного функціоналу для забезпечення ефективної комунікації, а також вибір відповідних засобів розробки. Для реалізації кросплатформеного рішення використано сучасні мови програмування та фреймворки, які дозволяють створювати гнучкі й масштабовані клієнтські застосунки. Важливим завданням є також забезпечення стабільної роботи системи в умовах нестабільного або обмеженого інтернет-з’єднання, що характерно для віддалених господарських точок аграрного виробництва.

Таке формулювання предметної області дозволяє чітко окреслити межі дослідження, його технічну спрямованість та визначити ключові аспекти, які впливають на архітектурні та функціональні рішення під час реалізації системи корпоративного зв’язку для аграрних підприємств.

11.5 Характеристика аграрних підприємств як користувачів корпоративного ПЗ

Аграрні підприємства є важливою складовою економіки України та активно інтегруються в процеси цифрової трансформації виробничих процесів. Їх характерними рисами є значна територіальна розподіленість персоналу, сезонність робіт, багаторівнева організаційна структура, а також постійна потреба в оперативному обміні інформацією між підрозділами. Це зумовлює високий попит на ефективні засоби внутрішньої комунікації, які сприяють оптимізації управлінських процесів, скороченню часових витрат на прийняття рішень та підвищенню загальної ефективності агровиробництва.

У типовій структурі аграрного підприємства функціонують адміністративні, агрономічні, технічні, логістичні та фінансові підрозділи, а також безпосередньо польові працівники. Усі ці категорії працівників здійснюють свою діяльність як в офісному середовищі, так і в польових умовах, що створює потребу в безперебійному каналі обміну повідомленнями, централізованому обліку виконаних дій, фіксації технічного стану об’єктів, а також у негайному отриманні розпоряджень незалежно від місця перебування.

Високі вимоги до інтерфейсу системи, її надійності в умовах слабкого інтернет-з’єднання, підтримки кількох платформ та захисту інформації є ключовими критеріями, які визначають придатність корпоративного програмного забезпечення для аграрного сектору. Простота використання є критичною умовою, оскільки не всі працівники мають досвід роботи з цифровими технологіями. Окрім цього, значну роль відіграє сезонність навантаження: під час посівної або жнив обсяг комунікації суттєво зростає, що вимагає від системи стабільності й здатності витримувати пікові навантаження без втрати повідомлень чи переривання роботи.

У сучасному корпоративному середовищі, особливо в аграрному, де працівники можуть бути розташовані на значній відстані один від одного — на полях, фермах, у складських чи офісних приміщеннях — важливою умовою

ефективної взаємодії є наявність надійного, безпечного і простого у використанні каналу комунікації. У зв’язку з цим корпоративні месенджери стали ключовими інструментами для підтримки зв’язку між підрозділами.

Месенджери забезпечують обмін текстовими повідомленнями, файлами, фото, відео та іншими типами даних у режимі реального часу. У корпоративному середовищі вони також виконують функцію координатора дій між різними службами, що дозволяє оперативно реагувати на зміну обставин, критичні ситуації або технічні виклики.

Сучасні месенджери, такі як Slack, Microsoft Teams або Telegram, уже пропонують базовий функціонал групових чатів, спільного використання файлів, а також підтримують голосовий та відеозв’язок. Вони мають високий рівень мультиплатформенності, дозволяючи користувачам використовувати їх на Windows, Linux, Android або iOS, що робить їх зручними у будь-якому середовищі.

Проте в аграрному контексті звичайні месенджери не завжди відповідають специфічним потребам. Для таких підприємств критично важливими є наявність офлайн-режиму роботи, можливість роботи з великими файлами у форматі peer-to-peer, а також централізоване адміністрування з гнучкою системою ролей та прав доступу. Саме ці аспекти враховуються при розробці корпоративного месенджера.

Проєктна реалізація включає використання сучасних засобів захисту інформації: end-to-end шифрування, автентифікацію користувачів за допомогою JWT-токенів, захищені SSL/TLS-з’єднання. Крім того, система підтримує кросплатформенну архітектуру та інтуїтивно зрозумілий інтерфейс, адаптований до користувачів без технічного досвіду.

Таким чином, на відміну від універсальних рішень, розроблений месенджер орієнтований саме на специфіку аграрного виробництва. Він забезпечує надійну, безпечну та гнучку комунікацію всередині підприємства, відповідає умовам польової роботи та сприяє ефективній координації дій усіх підрозділів незалежно від місця їх розташування.

11.6 Вимоги до корпоративного месенджера для аграрної сфери

Для забезпечення ефективної комунікації в умовах аграрного виробництва, корпоративний месенджер має відповідати цілій низці функціональних, технічних, безпекових, адміністративних та інтерфейсних вимог. Вони формуються на основі особливостей галузі, а також очікувань користувачів щодо зручності, стабільності та безпечності системи.

З функціональної точки зору, система повинна забезпечувати можливість обміну текстовими повідомленнями в режимі реального часу, підтримувати групові чати відповідно до організаційної структури підприємства (наприклад, «Агрономія», «Технічна служба», «Адміністрація») та дозволяти надсилання мультимедійних файлів, зокрема зображень, відео та аудіо. Додатковою важливою функцією є можливість прикріплення геолокації, що особливо актуально для польових умов. Також очікується наявність системи push-сповіщень для повідомлень критичного характеру та зберігання історії листування з локальним кешуванням для роботи в офлайн-режимі.

З технічного боку, месенджер має бути кросплатформеним і підтримувати роботу на різних операційних системах, зокрема Windows, Linux та Android. Оскільки польові умови часто супроводжуються нестабільним або обмеженим інтернет-з’єднанням, система повинна зберігати працездатність у таких випадках. Необхідна також можливість офлайн-доступу до історії повідомлень із подальшою автоматичною синхронізацією при відновленні з’єднання. Важливо, щоб клієнтська частина була оптимізована для економного споживання ресурсів пристрою, особливо енергії та пам’яті на мобільних пристроях.

Безпека корпоративної комунікації відіграє ключову роль. Месенджер повинен підтримувати шифрування повідомлень під час передачі з використанням SSL/TLS-протоколів, а також механізми автентифікації на

основі JWT або токенів доступу. Має бути передбачена можливість гнучкого налаштування прав доступу до каналів і чатів, захист від несанкціонованого доступу й підробки даних, а також створення журналів активності користувачів для моніторингу подій у системі.

З адміністративної точки зору, система повинна забезпечувати централізоване управління обліковими записами, з можливістю додавання або видалення користувачів та створення груп. Повинна існувати рольова модель доступу з поділом на адміністратора, модератора та звичайного користувача. Доцільним є впровадження автоматичного резервного копіювання даних, щоб забезпечити збереження інформації у випадку збоїв або втрати доступу.

Щодо інтерфейсу, месенджер має бути простим та інтуїтивно зрозумілим навіть для користувачів без технічної підготовки. Основні функції повинні виконуватись із мінімальною кількістю дій, а інтерфейс — підтримувати українську мову для зручності співробітників аграрного підприємства.

Таким чином, урахування всіх вищенаведених вимог дозволить створити ефективний, адаптований до реальних умов аграрної галузі програмний продукт, що підвищить оперативність прийняття рішень, зменшить ризики втрати інформації та сприятиме покращенню комунікації між структурними підрозділами підприємства.

11.7 Формування технічного завдання

Технічне завдання є ключовим документом, що визначає функціональні, технічні, організаційні та нефункціональні характеристики програмного забезпечення. Його формування базується на результатах аналізу предметної області, оцінці потреб користувачів, а також врахуванні умов експлуатації системи в аграрному середовищі. У межах цього дослідження було сформульовано технічне завдання на створення системи комунікації під назвою «Корпоративний месенджер для аграрних підприємств».

Метою створення системи є забезпечення ефективної, безпечної та зручної комунікації між співробітниками агропідприємства незалежно від їхнього місця перебування чи рівня технічної обізнаності. Месенджер має функціонувати в умовах польової роботи, обмеженого або нестабільного інтернет-з’єднання, а також відповідати принципам масштабованості та модульності.

Система повинна включати механізми реєстрації та автентифікації користувачів, які реалізуються за допомогою JSON Web Token (JWT). Вона повинна підтримувати обмін текстовими повідомленнями в режимі реального часу, а також дозволяти створення групових чатів з розмежуванням ролей і прав доступу. Користувачі повинні мати змогу надсилати зображення, відео та документи, а також отримувати push-сповіщення про нові повідомлення та події в системі. Важливою складовою є можливість зберігання історії повідомлень із функцією локального кешування для доступу в офлайн-режимі. Крім клієнтських застосунків, передбачається наявність вебінтерфейсу, що забезпечує базову взаємодію з системою через браузер.

Програмне забезпечення складається з кількох ключових компонентів. Клієнтська частина реалізована як кросплатформений застосунок на базі Qt/QML, з підтримкою ОС Windows, Linux та Android. Серверна частина створена з використанням мови програмування C++ та бібліотек Boost.Asio і Boost.Beast, які забезпечують реалізацію HTTP-сервера для обробки запитів. Сховище даних організоване на основі СУБД MySQL, яка зберігає інформацію про користувачів, повідомлення, групи та сесії. Для користувачів, які працюють у браузері, реалізовано вебінтерфейс за допомогою технологій HTML, CSS і JavaScript.

Нефункціональні вимоги до системи охоплюють підтримку роботи в умовах нестабільного інтернет-з’єднання, високу доступність, стійкість до відмов, простоту встановлення та оновлення клієнтських компонентів. Також передбачено захист усіх переданих даних шляхом використання протоколів

SSL/TLS, а для адміністраторів — логування подій і дій користувачів у системі.

Серед обмежень, що враховуються у ТЗ, визначено мінімально підтримувану версію Android — 7.0, обов’язкову сумісність із Windows 10 та Ubuntu 20.04 і вище. Передбачено автономність системи, без інтеграції з сторонніми месенджерами на кшталт Telegram або WhatsApp, що дозволяє зберігати повний контроль над внутрішніми процесами підприємства.

Очікуваним результатом виконання технічного завдання є створення повноцінної клієнт-серверної системи, здатної забезпечити надійне інформаційне середовище для комунікації між польовими та офісними працівниками аграрного підприємства, з урахуванням можливості масштабування, розширення функціональності та адаптації до потреб конкретної організаційної структури.

11.8 Діаграми прецедентів і послідовності

Для формалізації вимог до програмної системи та моделювання поведінки користувачів щодо взаємодії з корпоративним месенджером було побудовано діаграми прецедентів та діаграми послідовностей на основі мови моделювання UML.

Діаграма прецедентів на рис. 1 відображає основні дії, які можуть виконувати користувачі системи залежно від їх ролей. У системі передбачено три типи акторів: користувач (працівник), адміністратор, система авторизації.

Кожна дія пов'язана з попередньою перевіркою прав доступу користувача. Взаємодія між прецедентами реалізована через відношення <<include>>, що вказує на обов’язковість виконання супровідних дій (наприклад, перевірка ролі перед зміною даних).

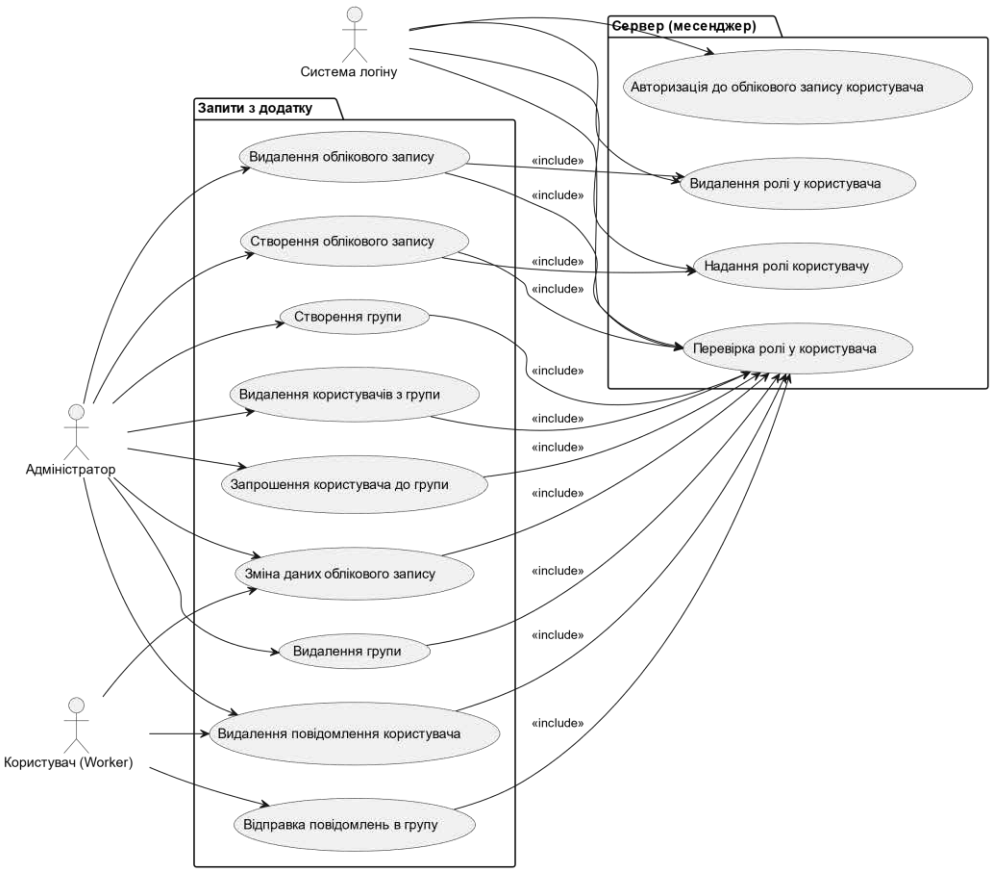


Рис. 1 Діаграма прецедентів месенджеру

Діаграми послідовностей відображають логіку обміну повідомленнями та дій користувачів під час взаємодії з системою, зокрема з веб-версією та десктопним клієнтом.

На діаграмі зображеної на рис. 2 продемонстровано процес перегляду чатів і відправлення повідомлення користувачем у клієнтському застосунку. Користувач відкриває чат, отримує поточну інформацію від сервера. Після створення повідомлення клієнт надсилає його на сервер, де воно зберігається й підтверджується у відповідь. Повідомлення відображається в інтерфейсі, а чат оновлюється.

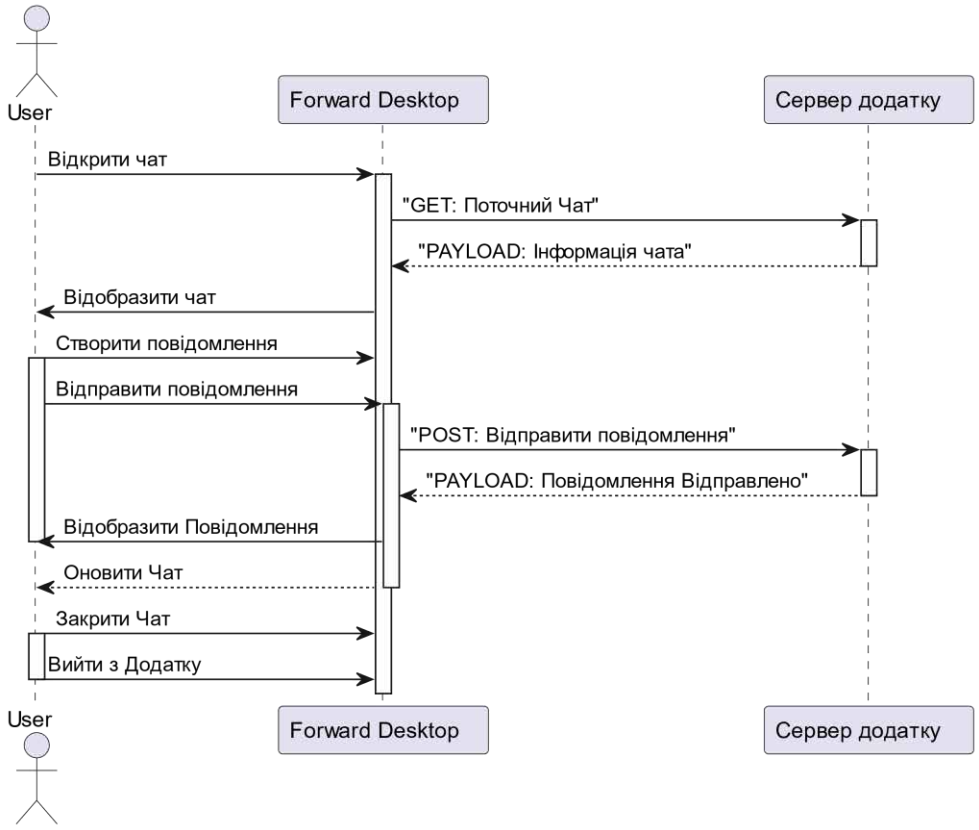


Рис. 2 Діаграма послідовності «Перегляд і відправлення повідомлення у клієнтському застосунку»

На рис. 3 зображено процес реєстрації нового користувача через веб-інтерфейсю Спочатку, користувач відкриває сайт і форму створення облікового запису. Після заповнення й відправлення форми дані зберігаються в базі. Потім, користувач отримує підтвердження створення і переходить до логін-форми, після чого відбувається перевірка облікових даних.

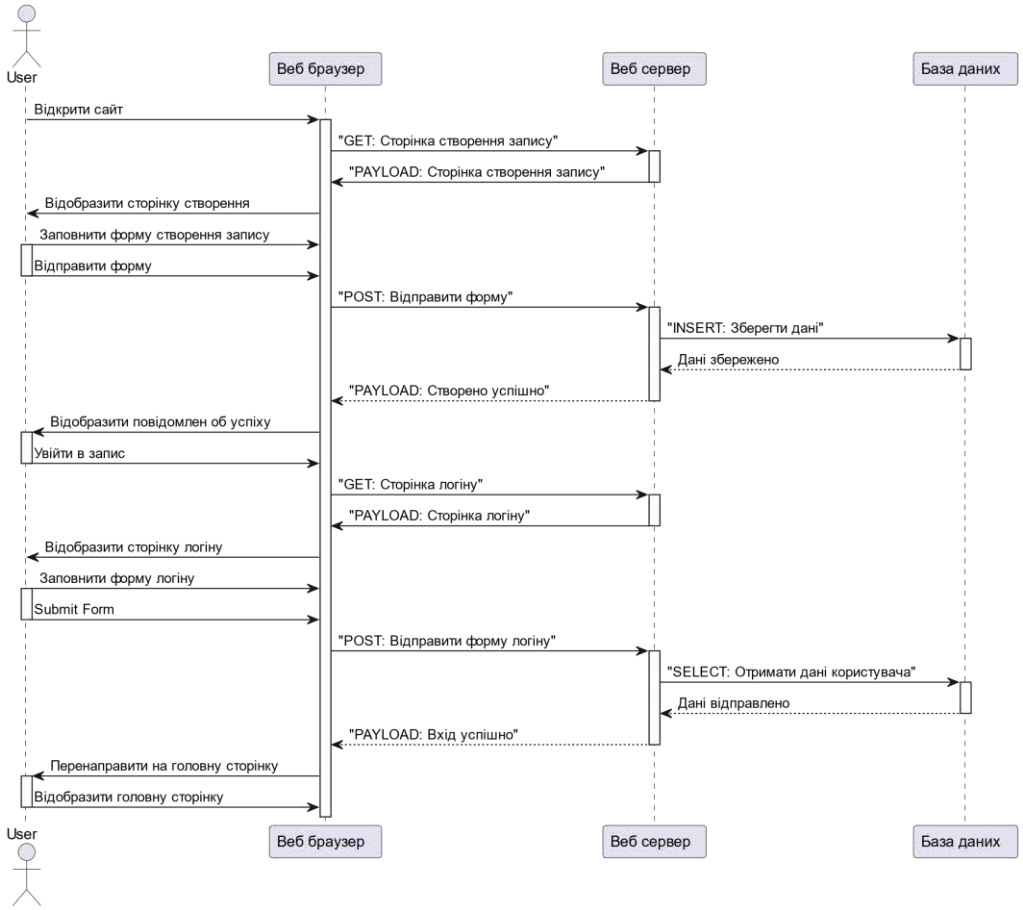


Рис. 3 Діаграма послідовності «Реєстрація та вхід користувача через веб-інтерфейс»

21 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

12.1 Вибір методів та засобів розробки

Вибір методів та засобів для реалізації прикладного програмного забезпечення для проєкту було здійснено з урахуванням кількох факторів. Основними критеріями при виборі методів та засобів є ефективність, зручність використання, масштабованість та підтримка необхідних функцій.

У розробці месенджеру для аграрних підприємств важливим аспектом є належне управління процесом збирання проєкту. Один з найпопулярніших і потужних інструментів для автоматизації цього процесу є CMake. CMake (див. рис. 4) є крос-платформовим інструментом для генерації скриптів збирання, який дозволяє розробникам описувати структуру проєкту та його залежності.

Рис. 4 Логотип CMake

Для розробки серверної частини месенджера обрано мову програмування C++. Вибір цієї мови обумовлений її високою продуктивністю та можливістю оптимізації, що дозволяє досягти високої швидкодії роботи програми [5]. Крім того, C++ надає доступ до потужних бібліотек, як Boost.Asio та Boost.Beast (див. рис. 5).

Рис. 5 Логотип Boost C++

Boost.Asio [6] є бібліотекою для асинхронного програмування вводу/виводу (I/O) в C++. Вона надає розширений функціонал для роботи з сокетами, мережевими протоколами, серійним портом, а також підтримку асинхронних операцій вводу/виводу та таймерів. Boost.Asio дозволяє створювати асинхронні програми, які здатні ефективно обробляти багато з'єднань одночасно, виконувати мережеві операції без блокування та забезпечувати гнучкість управління подіями вводу/виводу.

Boost.Beast є бібліотекою для маніпулювання мережевими протоколами в середовищі C++. Вона надає високорівневий доступ до протоколів HTTP, WebSocket і Networking TS. Boost.Beast дозволяє зручно створювати клієнтські та серверні програми, обмінюватися даними через HTTP або WebSocket, і здійснювати роботу з різними частинами веб-додатків, такими як URL, заголовки, тіла повідомлень і т.д. Вона також надає підтримку асинхронної роботи з мережевими операціями.

Використання OpenSSL, (логотип див. рис. 6) дозволяє реалізувати шифрування та розшифрування повідомлень, підписування та перевірку цифрових підписів, а також створення безпечних каналів зв'язку за допомогою протоколів, таких як Transport Layer Security (TLS) та Secure Sockets Layer (SSL) [7]. Це дозволяє забезпечити конфіденційність, цілісність та автентичність даних, переданих через месенджер.

Рис. 6 Бібліотека шифрування OpenSSL

OpenSSL надає широкий спектр криптографічних алгоритмів, включаючи симетричні та асиметричні алгоритми шифрування, хеш-функції, алгоритми генерації ключів та багато іншого. Використання цих алгоритмів дозволяє забезпечити безпеку обміну повідомленнями та захист інформації від несанкціонованого доступу.

Додавання підтримки OpenSSL до проєкту вимагає налаштування та інтеграції бібліотеки в додаток. Це включає встановлення OpenSSL на сервері, який використовується для обробки повідомлень, і реалізацію криптографічних функцій у вихідному коді месенджера.

Ці бібліотеки дозволяють забезпечити функціонал HTTPS та WebSocket серверів, що дозволяють здійснювати зв'язок у реальному часі з клієнтською частиною месенджера. Крім того, використовується власна бібліотека ForwardAPI, яка допомагає реалізувати швидкий та ефективний обмін даними між сервером та клієнтом.

HTTPS (Hypertext Transfer Protocol Secure) - це захищений протокол передачі гіпертексту, що використовується для безпечної комунікації між клієнтом і сервером через мережу Інтернет. HTTPS є розширенням стандартного протоколу HTTP [7] і забезпечує шифрування даних, аутентифікацію сервера та цілісність повідомлень. Наглядний приклад роботи протокола зображено на рис. 7.

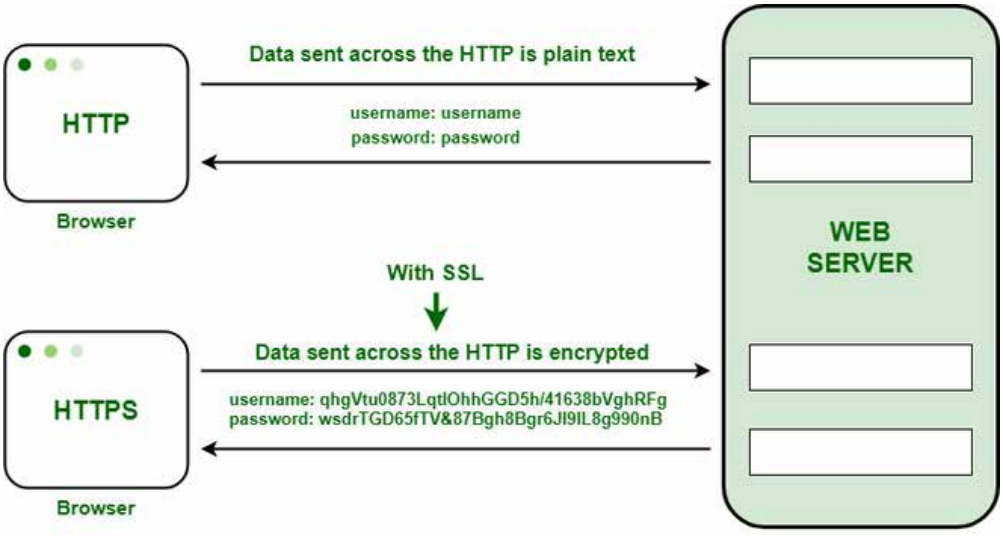


Рис. 7 Робота SSL підключення

TLS (Transport Layer Security) — це сучасний криптографічний протокол, який забезпечує захист даних на транспортному рівні під час їх передавання мережею. Відповідно до опису, поданого в роботі [7], протокол TLS забезпечує три ключові аспекти безпеки інформаційного обміну.

Перш за все, TLS гарантує шифрування даних, що унеможливлює їх перехоплення або прочитання третіми особами. Завдяки цьому навіть у разі втручання зловмисника, зміст інформації залишиться недоступним, оскільки він зашифрований та не може бути дешифрований без відповідного ключа.

Другою важливою властивістю є забезпечення цілісності даних, що означає захист інформації від спотворення під час передачі. Усі зміни, внесені до даних у процесі транспортування — навмисно чи випадково — будуть виявлені, а отже, система зможе відреагувати відповідним чином, наприклад, відхилити пошкоджений пакет даних.

Третій елемент, який забезпечує TLS, — це аутентифікація сторін. Завдяки механізму перевірки автентичності, користувач отримує гарантію, що з'єднання встановлено саме з тією стороною, з якою він мав намір взаємодіяти. Це дозволяє уникнути атак типу "людина посередині" (MITM) і створює основу для формування довіри до сайту або сервісу з боку кінцевого користувача.

У контексті корпоративного месенджера така система забезпечення безпеки є критично важливою, оскільки дозволяє зберегти конфіденційність повідомлень, гарантувати їх достовірність та захистити корпоративні дані від потенційних загроз у мережевому середовищі.

Клієнтської частини було обрано мову програмування C++ та фреймворк Qt та мову розмітки QML. Qt є потужним інструментом для розробки кросплатформних додатків [3], що дозволяє швидко створювати графічний інтерфейс користувача та взаємодіяти з різними функціями додатку. QtFramework дозволяє створювати переносимі та інтерактивні графічні інтерфейси користувача. Використання QML дозволяє зручно розробляти інтерактивні елементи користувацького інтерфейсу. Клієнтська частина також використовує ForwardAPI для взаємодії з серверною частиною.

Для доступу до месенджера також створений веб-сайт, який розроблений з використанням HTML, CSS та JavaScript [8]. Веб-сайт дозволяє

користувачам зареєструватись та завантажити додаток. Він надає зручний інтерфейс для входу, реєстрації.

Проєкт використовує базу даних MySQL для зберігання інформації про користувачів, повідомлення та інші додаткові дані, необхідні для роботи месенджера. Вона забезпечує надійне зберігання та організацію даних для швидкого доступу під час обміну повідомленнями.

База даних, що використовується у корпоративному месенджері, виконує функцію централізованого сховища інформації, яка забезпечує коректну роботу всієї системи. Вона містить повний набір даних, необхідних для автентифікації користувачів, зберігання повідомлень, управління контактами, налаштуваннями профілю та обробки групових чатів.

Однією з основних частин бази є інформація про зареєстрованих користувачів. Для кожного користувача зберігаються унікальний ідентифікатор, ім’я, контактні дані (зокрема електронна пошта або номер телефону), захищений пароль, а також додаткові елементи профілю, зокрема коротке біо або аватар. Ці дані є базовими для забезпечення персоналізованого доступу до системи та ідентифікації користувача при взаємодії з іншими учасниками.

Окремий модуль бази відповідає за зберігання усіх повідомлень, які обмінюються між користувачами. Для кожного повідомлення фіксується його унікальний ідентифікатор, відправник, отримувач або група, зміст повідомлення, час надсилання, а також статус перегляду. Це дозволяє організовувати синхронізацію листування, реалізовувати функцію перегляду історії та зберігати цілісність комунікації.

Система також підтримує зберігання інформації про соціальні зв’язки між користувачами — їхні списки друзів та контактів. Це дозволяє швидко знаходити потрібного співрозмовника, формувати групи за інтересами або підрозділами підприємства, а також реалізовувати функції запрошень та блокування.

Ще одним важливим компонентом є налаштування профілю. Для кожного користувача в базі зберігаються його індивідуальні параметри: налаштування сповіщень, теми оформлення інтерфейсу, мова системи, дозволи на прийом повідомлень від незнайомих користувачів тощо. Це забезпечує персоналізований досвід використання системи.

Інформація про групи та чати також структурується в окремих таблицях. Тут фіксується назва групи, склад учасників, дата створення, роль кожного учасника та інші адміністративні параметри. Кожна група пов’язана з повідомленнями, надісланими в її межах, що дозволяє зберігати повноцінну історію спілкування.

Крім звичайних повідомлень, база даних може зберігати й системні повідомлення. Вони призначені для інформування користувачів про критичні події, помилки, оновлення або зміну політик безпеки. Ці записи є важливими для ведення журналів подій і підтримки прозорості взаємодії системи з користувачем.

Ці дані в базі даних допоможуть забезпечити зручну та надійну роботу месенджера, зберігаючи необхідну інформацію для комунікації та налаштування користувачів.

Сесії JWT (JSON Web Token) можна також включити в проєкт. Приклад структури токенів зображено на рис. 8.

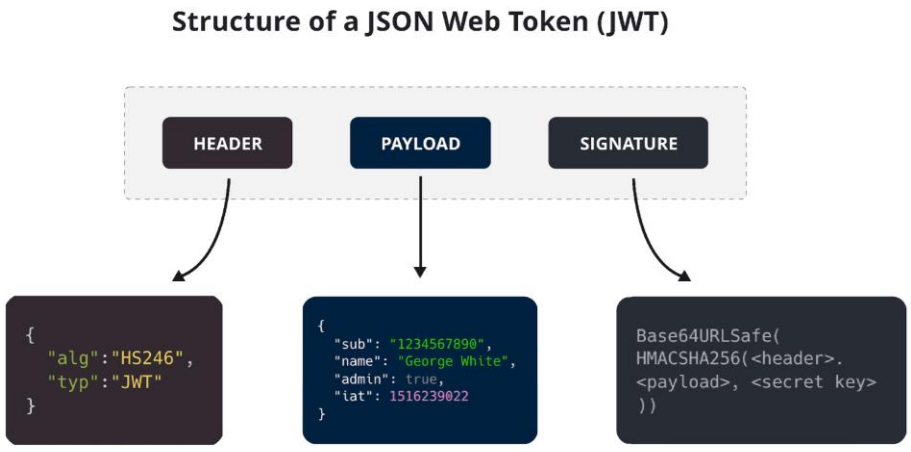


Рис. 8 Представлення структури JSON Web Token

JSON Web токени мають певний строк дії, після якого вони стають недійсними. Це дозволяє контролювати тривалість сесій користувачів.

Крім того, JWT токени можуть містити інформацію про ролі користувачів, які надаються права доступу до різних ресурсів месенджера.

Використання сесій JWT у месенджері допоможе забезпечити безпеку та контроль доступу до ресурсів додатку, дозволяючи ефективно управляти авторизацією користувачів та їх сесіями. Токени будуть зберігатися на стороні клієнта, що спрощує процес управління та зменшує навантаження на серверну частину додатку.

12.2 Вибір домену для веб-сайту

Для веб-сайту месенджеру був обраний домен за допомогою сервісу доменних імен GoDaddy див. рис. 9. Домен, який був вибраний, є називається – forchat.online.

Рис. 9 Логотип сайту GoDaddy

Обираючи домен, було враховано кілька факторів. Важливим було мати короткий та запам'ятовуваний домен, що легко асоціюється з ідеєю проєкту. Домен “forchat.online” передає суть проєкту, оскільки слово “forchat” вказує на можливість спілкування, а розширення “online” вказує на наявність веб-присутності.

Обраний домен “forchat.online” відповідає потребам «Розробки корпоративного месенджера для аграрних підприємств».

Сервіс GoDaddy був обраний для придбання домену через свою високу репутацію та надійність. GoDaddy є одним з найбільших і провідних провайдерів доменних імен та хостингу в Інтернеті. Компанія була заснована в 1997 році і швидко стала популярною завдяки своїй широкій набору послуг і простому процесу реєстрації доменних імен.

GoDaddy надає можливість створювати власні електронні адреси з вашим доменом. Ви можете налаштувати поштові скриньки і використовувати їх для професійної комунікації.

Крім реєстрації доменів і хостингу, GoDaddy також пропонує інструменти для створення веб-сайтів. Вони мають власний конструктор веб-сайтів, який дозволяє легко створювати професійні сайти без необхідності програмування.

GoDaddy пропонує SSL-сертифікати, які дозволяють зашифровувати передачу даних між веб-сайтом і його відвідувачами. Це забезпечує безпеку і довіру для користувачів вашого сайту.

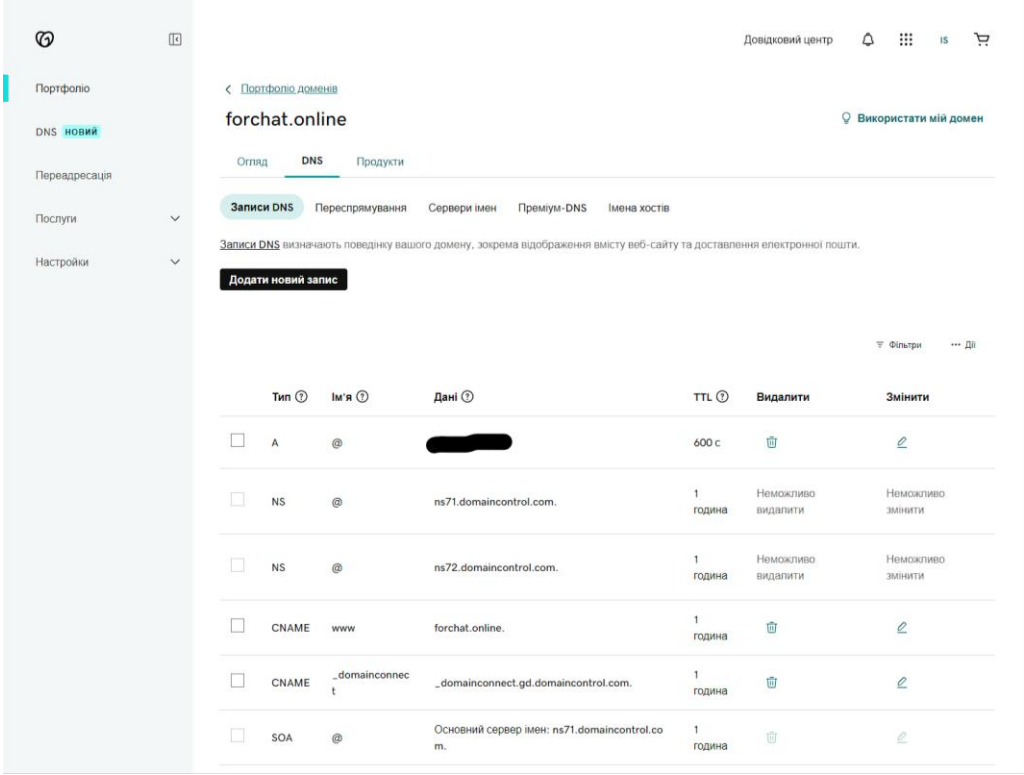


Рис. 10 Панель управління доменом

Одним із переваг використання GoDaddy є їхній простий інтерфейс Рис. 10, який робить процес реєстрації домену та налаштування послуг легким і доступним для користувачів. Вони також надають інструменти для управління доменами, DNS-записами та іншими параметрами в одному місці.

12.3 Інформаційне забезпечення месенджеру

Інформаційне забезпечення є важливою складовою будь-якої інформаційної системи, зокрема корпоративного месенджера. Воно охоплює методи, засоби та структури, які забезпечують накопичення, зберігання, обробку та доступ до даних, необхідних для функціонування програмного забезпечення.

У якості системи управління базами даних у корпоративному месенджері використовується MySQL — одна з найпопулярніших реляційних СУБД з відкритим кодом. Вона забезпечує високу продуктивність, масштабованість, надійність і підтримує використання мови структурованих запитів SQL (Structured Query Language), яка є стандартом для роботи з реляційними даними.

MySQL дозволяє зберігати велику кількість взаємопов'язаних структурованих даних, що необхідно для реалізації функціоналу месенджера: облік користувачів, історія листування, структура контактів, групові чати, мультимедійні вкладення, дані про сесії, пристрої, профілі тощо. Інформація розподіляється між взаємопов’язаними таблицями бази даних, що забезпечує цілісність, узгодженість та захищеність даних.

У межах інформаційного забезпечення система включає механізми контролю доступу до даних, забезпечення цілісності транзакцій, індексування для пришвидшення пошуку, а також можливість журналювання змін для подальшого аналізу чи аудиту. Забезпечується підтримка одночасного доступу багатьох клієнтів до даних із дотриманням принципів багатокористувацької роботи та ізоляції транзакцій.

Крім цього, інформаційне забезпечення включає зберігання службових і системних даних, таких як налаштування користувачів, логування активностей, шаблони сповіщень та журнал помилок, що сприяє стабільній роботі та адмініструванню системи.

Таким чином, інформаційне забезпечення на базі MySQL виконує ключову роль у функціонуванні корпоративного месенджера, забезпечуючи надійне середовище для обміну, збереження та обробки комунікаційних даних, необхідних для повноцінної роботи аграрного підприємства.

12.4 Логічна модель даних

Діаграма бази даних (Database Schema Diagram) є візуальним засобом представлення логічної структури бази даних. Вона використовується для графічного зображення таблиць, їх атрибутів, а також зв’язків між ними. Така діаграма дає змогу наочно уявити, як саме організовано зберігання даних у системі, які об’єкти взаємодіють між собою та які залежності між ними встановлені.

Кожна таблиця в такій діаграмі, як правило, представлена у вигляді прямокутника, що містить її назву. Усередині таблиці вказуються всі колонки (атрибути) з відповідними назвами. Зв’язки між таблицями, такі як «один до багатьох» або «багато до багатьох», зображуються лініями, що з’єднують відповідні таблиці й підкреслюють логіку їхньої взаємодії. Особливу увагу приділяють ключам: первинні ключі позначають унікальність записів у таблиці, а зовнішні ключі — демонструють, які саме поля пов’язують цю таблицю з іншими.

Діаграма бази даних є важливим інструментом під час проєктування інформаційної системи, оскільки забезпечує цілісне бачення структури зберігання даних, дозволяє виявити логічні помилки на ранньому етапі та сприяє ефективній комунікації між розробниками, аналітиками та адміністраторами баз даних.

Діаграма бази даних допомагає розробникам, аналітикам і адміністраторам баз даних краще зрозуміти структуру даних, зв'язки між таблицями та розміщення атрибутів. Вона також може служити документацією для проєкту та допомагати при аналізі, проєктуванні та оптимізації бази даних.

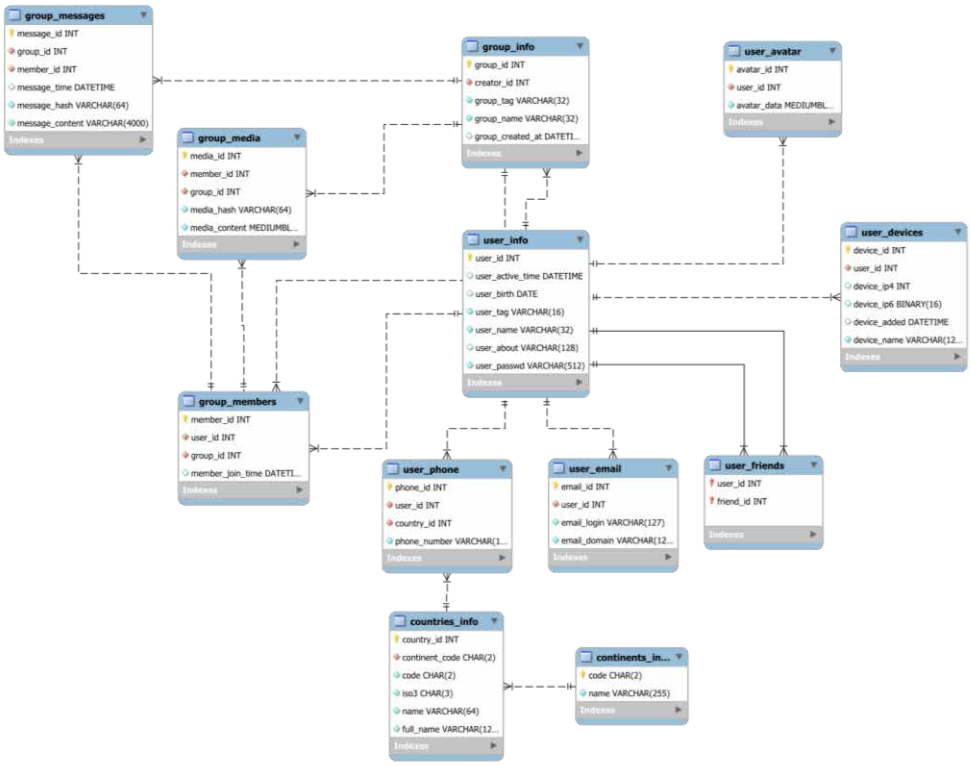


Рис. 11 ER Діаграма бази даних месенджеру

На рис. 11 зображено структуру бази даних месенджеру. Вона включає в себе сутності для користувачів, груп, повідомлень, контактів.

31 РОЗРОБКА КОРПОРАТИВНОГО МЕСЕНДЖЕРА

13.1 Розробка бази даних

Виконнання запиту для створення бази даних відбувається за допомогою команди CREATE DATABASE. Реалізація даної команди та безпосереднє створення бази даних див. рис. 12.

```
CREATE database forward-db;
```

Рис. 12 Створення бази даних месенджеру

13.1.1 Створення таблиць

Таблиці — це набір елементів даних, які організовані з використанням моделі вертикальних стовпчиків з різними іменами та горизонтальних рядків.

Створюються таблиці за допомогою команди CREATE TABLE. Для створення необхідно вказати: ім’я таблиці, яке вказується після ключового слова CREATE TABLE; назви та тип даних стовпців таблиці, що відділені комами; додаткові оператори такі як: NOT NULL (не може бути пустим), ключові поля, авто-заповнюючи поля (код, дата, час).

Щоб зберігати дані користувача, створено таблицю див. рис. 13. Яка зберігає номер, унікальну назву, ім’я, інформацію та хешований зі сіллю (зміщенням) пароль.

```
CREATE TABLE `user_info` (
`user_id` int NOT NULL AUTO_INCREMENT,
`user_tag` varchar(32) CHARACTER SET ascii COLLATE ascii_general_ci NOT NULL,
`user_name` varchar(32) CHARACTER SET utf16 COLLATE utf16_general_ci NOT NULL,
`user_pass_hash` varchar(256) CHARACTER SET ascii COLLATE ascii_general_ci NOT NULL,
`user_about` varchar(100) CHARACTER SET utf16 COLLATE utf16_general_ci DEFAULT NULL,
`user_creation` date DEFAULT NULL,
`last_online` datetime DEFAULT NULL,
PRIMARY KEY (`user_id`),
UNIQUE KEY `user_tag_UNIQUE` (`user_tag`))
ENGINE=InnoDB
AUTO_INCREMENT=3
DEFAULT CHARSET=utf8mb3
```

Рис. 13 Табилця user_info

Для користувача також потрібно реалізувати таблицю профіля, як зображено на рис. 14. Де будуть зберагітись картинки профілю користувача. Також користувач може вказати яка картинка буде відображатись головною.

```
CREATE TABLE `user_avatar` (
  `avatar_id` int NOT NULL,
  `user_id` int NOT NULL,
  `avatar_data` mediumblob NOT NULL,
  `is_selected` tinyint DEFAULT '1',
  PRIMARY KEY (`avatar_id`),
  KEY `user_id_idx` (`user_id`),
  CONSTRAINT `fk_avatar_user_id`
  FOREIGN KEY (`user_id`) REFERENCES `user_info` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3
```

Рис. 14 Таблиця user_avatar

Для відновлення паролю та входу за допомогою пошти створено таблицю «user_email» див. рис. 15, яка зберігає пошту користувача.

```
CREATE TABLE `user_email` (
  `email_id` int NOT NULL,
  `user_id` int NOT NULL,
  `email_adress` varchar(255) NOT NULL,
  PRIMARY KEY (`email_id`),
  KEY `fk_user_email_user_id_idx` (`user_id`),
  CONSTRAINT `fk_user_email_user_id`
  FOREIGN KEY (`user_id`) REFERENCES `user_info` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=ascii
```

Рис. 15 Таблиця user_email

Для відновлення паролю та входу за допомогою номера телефона було створено таблицю «user_phone» як на рис. 16, яка зберігає номер телефону, країну користувача.

```
CREATE TABLE `user_phone` (
  `phone_id` int NOT NULL AUTO_INCREMENT,
  `user_id` int NOT NULL,
  `phone_number` varchar(45) NOT NULL,
  `phone_country` varchar(3) NOT NULL,
  PRIMARY KEY (`phone_id`),
  KEY `fk_user_phone_user_id_idx` (`user_id`),
  CONSTRAINT `fk_user_phone_user_id`
  FOREIGN KEY (`user_id`) REFERENCES `user_info` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=ascii
```

Рис. 16 Таблиця user_phone

Наступна таблиця «user_friends» див. рис. 17, вона зберігає номери користувача та друга. Також в дану таблицю було додано унікальні поля які не можуть повторюватись.

```
CREATE TABLE `user_friends` (
  `friend_id` int NOT NULL AUTO_INCREMENT,
  `user_id` int NOT NULL,
  `user_friend_id` int NOT NULL,
  PRIMARY KEY (`friend_id`),
  UNIQUE KEY `uc_user_friends` (`user_id`,`user_friend_id`),
  CONSTRAINT `fk_user_friends_user_id` FOREIGN KEY (`user_id`) REFERENCES `user_info` (`user_id`),
  CONSTRAINT `chk_user_friend_different` CHECK ((`user_id` <> `user_friend_id`))
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb3
```

Рис. 17 Таблиця user_friends

Щоб API серверу розумів чи є користувачі в групі, було створено допоміжну таблицю для зберігання номерів групи та користувача.

```
CREATE TABLE `user_groups` (
  `user_group_id` int NOT NULL AUTO_INCREMENT,
  `user_id` int NOT NULL,
  `group_id` int NOT NULL,
  PRIMARY KEY (`user_group_id`),
  KEY `fk_user_group_user_id_idx` (`user_id`),
  KEY `fk_user_group_group_id_idx` (`group_id`),
  CONSTRAINT `fk_user_group_group_id`
  FOREIGN KEY (`group_id`) REFERENCES `group_info` (`group_id`),
  CONSTRAINT `fk_user_group_user_id`
  FOREIGN KEY (`user_id`) REFERENCES `user_info` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf16
```

Рис. 18 Таблиця user_groups

Також створено таблицю «group_info» рис. 19 для групових співбесіди, яка зберігає назву, картинку, дату створення та самого користувача який створив цю групу.

```
CREATE TABLE `group_info` (
  `group_id` int NOT NULL AUTO_INCREMENT,
  `user_id` int NOT NULL,
  `group_name` varchar(32) NOT NULL,
  `creation_time` date NOT NULL,
  `group_img` mediumblob,
  PRIMARY KEY (`group_id`),
  KEY `fk_group_info_user_id_idx` (`user_id`),
  CONSTRAINT `fk_group_info_user_id`
  FOREIGN KEY (`user_id`) REFERENCES `user_info` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3
```

Рис. 19 Таблиця group_info

Для зберігання медіа контенту було створено таблицю «group_media», зображено на рис. 20. Таблиця налічує такі поля як: номер медіа, номер групи, поле для зберігання медіа файлів, тип файлу, хеш-сума.

```
CREATE TABLE `group_media` (
  `media_id` int NOT NULL AUTO_INCREMENT,
  `group_id` int NOT NULL,
  `media_content` mediumblob NOT NULL,
  `media_hash` varchar(45) CHARACTER SET utf16 NOT NULL,
  `mime_type` varchar(45) CHARACTER SET utf16 NOT NULL,
  PRIMARY KEY (`media_id`),
  KEY `fk_group_media_idx` (`group_id`),
  CONSTRAINT `fk_group_media_group_id`
  FOREIGN KEY (`group_id`) REFERENCES `group_info` (`group_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
```

Рис. 20 Таблиця group_media

Для зберігання повідомлень було створено таблицю «group_messages», приклад на рис. 21, яка дозволяє зберігати текстові повідомлення до 4000 символів. Також вона створює два ключі на номери груп та користувачів.

```
CREATE TABLE `group_messages` (
  `message_id` int NOT NULL AUTO_INCREMENT,
  `group_id` int NOT NULL,
  `user_id` int NOT NULL,
  `message_content` varchar(4000) NOT NULL,
  `sent_date` datetime NOT NULL,
  PRIMARY KEY (`message_id`),
  KEY `fk_group_messages_user_id_idx` (`user_id`),
  KEY `fk_group_messages_group_id` (`group_id`),
  CONSTRAINT `fk_group_messages_group_id`
  FOREIGN KEY (`group_id`) REFERENCES `group_info` (`group_id`),
  CONSTRAINT `fk_group_messages_user_id`
  FOREIGN KEY (`user_id`) REFERENCES `user_info` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf16;
```

Рис. 21 Таблиця group_messages

Розроблена таблиця доступних чатів, як на рис. 22, для користувача яка має номери чату, користувача та друга.

```
CREATE TABLE `user_chats` (
  `chat_id` int NOT NULL AUTO_INCREMENT,
  `user_id` int NOT NULL,
  `friend_id` int NOT NULL,
  PRIMARY KEY (`chat_id`),
  KEY `fk_user_chats_friend_id_idx` (`friend_id`),
  KEY `fk_user_chats_user_id_idx` (`user_id`),
  CONSTRAINT `fk_user_chats_friend_id`
  FOREIGN KEY (`friend_id`) REFERENCES `user_info` (`user_id`),
  CONSTRAINT `fk_user_chats_user_id`
  FOREIGN KEY (`user_id`) REFERENCES `user_info` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf16
```

Рис. 22 Таблиця user_chats

Для зберігання повідомлень чатів було створено таблицю «chat_messages», зображено на рис. 23, яка дозволяє зберігати текстові повідомлення до 4000 символів. Також вона створює два ключі на номери чатів та користувачів.

```
CREATE TABLE `chat_messages` (
  `message_id` int NOT NULL AUTO_INCREMENT,
  `chat_id` int NOT NULL,
  `sender_id` int NOT NULL,
  `message_content` varchar(4000) NOT NULL,
  `chat_type` varchar(64) NOT NULL,
  `sent_date` datetime NOT NULL,
  `message_hash` varchar(256) DEFAULT NULL,
  PRIMARY KEY (`message_id`),
  KEY `fk_chat_messages_chat_id_idx` (`chat_id`),
  KEY `fk_chat_messages_sender_id_idx` (`sender_id`),
  CONSTRAINT `fk_chat_messages_chat_id`
  FOREIGN KEY (`chat_id`) REFERENCES `user_chats` (`chat_id`),
  CONSTRAINT `fk_chat_messages_sender_id`
  FOREIGN KEY (`sender_id`) REFERENCES `user_info` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf16;
```

Рис. 23 Таблиця chat_messages

Розроблюючи месенджер, також потрібно враховувати паралельні сессії. Створивши таблицю девайсів, яка буде відображати кожен девайс який зайшов в обліковий запис. Для таблиці знадобиться створити такі поля: номер, назва, хеш-сума та поточний девайс. На рис. 24 зображено приклад для створення запиту даної таблиці.

```
CREATE TABLE `user_devices` (
  `device_id` int NOT NULL AUTO_INCREMENT,
  `user_id` int NOT NULL,
  `device_name` varchar(128) NOT NULL,
  `is_current` tinyint DEFAULT '0',
  PRIMARY KEY (`device_id`),
  KEY `fk_user_devices_user_id_idx` (`user_id`),
  CONSTRAINT `fk_user_devices_user_id`
  FOREIGN KEY (`user_id`) REFERENCES `user_info` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=ascii
```

Рис. 24 Табилця user_devices

13.1.2 Створення представлень (views)

У системі корпоративного месенджера створення представлень (views) [9] є важливим етапом логічної організації доступу до даних. Представлення дозволяють спростити взаємодію з базою даних, приховати складну логіку

запитів, а також забезпечити рівень абстракції для зовнішніх систем або адміністративного інтерфейсу.

Однією з таких є аналітика онлайну користувачів «active_users» яка надає звіт. Приклад реалізації представлення на рис. 25.

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `active_users` AS
    SELECT
        `user_info`.`user_id` AS `user_id`
    FROM
        `user_info`
    WHERE
        (`user_info`.`user_active_time` >= (NOW() - INTERVAL 30 SECOND))
```

Рис. 25 Представлення кількості активних користувачів в системі

13.1.3 Створення функцій

Для атвоматизації деякий процесів в базі даних також було створено декілька функцій яки спрощують доступ до даних [10].

Одна з таких функцій називається «is_user_active» (рис. 26), яка має один аргумент «id» користувача. Дана функція повертає онлайн статус користувача в системі.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `is_user_active`(id INT) RETURNS tinyint(1)
    READS SQL DATA
    DETERMINISTIC
BEGIN
    DECLARE result BOOLEAN;
SELECT
    EXISTS( SELECT
            1
        FROM
            active_users
        WHERE
            user_id = id)
INTO result;
    RETURN result;
END
```

Рис. 26 Функція онлайн статусу користувача

Також було створено функцію утилиту яка повертає тег користувача з «id» користувача під назвою «get_user_tag» див. рис. 27.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `get_user_tag`(id INT) RETURNS varchar(16)
    READS SQL DATA
    DETERMINISTIC
BEGIN
DECLARE result VARCHAR(16);
SELECT
    user_info.user_tag
INTO result FROM
    user_info
WHERE
    user_info.user_id = id LIMIT 1;
RETURN result;
END
```

Рис. 27 Функція конвертації id до тегу

13.1.4 Створення користувачів

Кожен сервер має обмежений доступ до баз даних, щоб зменшити ризик зловмисникам мати повний контроль над сервером MySQL. Тому було додано ролі для того щоб сервер мав обмежені права. В них будуть входити: додавання, оновлення, вивід полів в таблицях і тільки це. Ролей повинно бути багато під кожного працівника або сервер. Перейдемо до створення ролей в MySQL.

Щоб надати роль, потрібно використати команду GRANT та вписати список команд які будуть доступні для ролі. В нашому випадку ми передаємо туди параметри: SELECT, INSERT, UPDATE. Після цього ми вказуємо ON - це буде вказувати що передаєм дані дії на об'єкт, ми обираємо таблиці (TABLE). Кінцевий результат буде виглядати як на рис. 28.

```
GRANT SELECT, INSERT, UPDATE
ON TABLE TO forward-server;
```

Рис. 28 Запит на надання дій ролі

Коли ми створили ролі як на рис. 29, наша база даних прийматиме тільки їх з вказаним паролем та ip адресою.

User	From Host
forward-api	%
forward-server	%

Рис. 29 Створені користувачі

13.2 Створення сценаріїв проєкту за домогою CMake

У розробці програмного забезпечення важливим аспектом є належне управління процесом збирання проєкту. Один з найпопулярніших і потужних інструментів для автоматизації цього процесу є CMake див. рис. 30. CMake є крос-платформовим інструментом для генерації скриптів збирання, який дозволяє розробникам описувати структуру проєкту та його залежності.

Рис. 30 Логотип CMake

Проєкт повністю збудований на даному генераторі. Приклад структури файлів серверу месенджера на рис. 31. На даному рисунку зображено фалову структуру проекту яку потрібно сконвертувати до скриптів CMake.

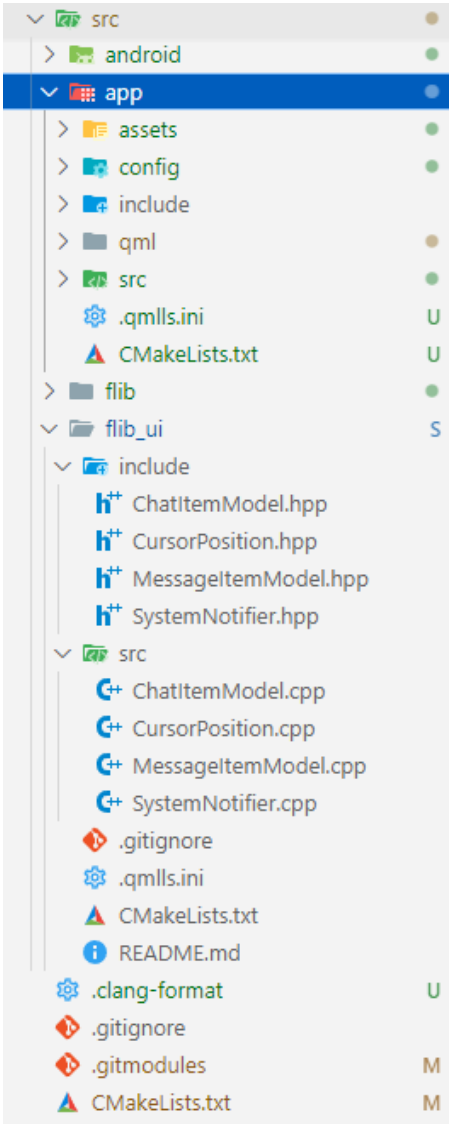


Рис. 31 Файлова структура додатку

На рис. 32 зображено приклад перенесеної структури файлів до CMake [11]. Дані «списки» описують з якою конфігурацією проєкт буде зібраний.

```
cmake_minimum_required( VERSION 3.15 )

set( CMAKE_CXX_STANDARD 17 )
set( CMAKE_CXX_STANDARD_REQUIRED ON )

project( ForwardDesktop
  VERSION 0.1
  LANGUAGES CXX
  DESCRIPTION "Forward Desktop Client based on Qt"
)

set( FORWARD_URI "chat.forward.ua" )
set( FORWARD_DOMAIN "forchat.online" )

set( FORWARD_APP_NAME "Messenger Forward" )
set( FORWARD_APP_VERSION "${PROJECT_VERSION_MAJOR}.${PROJECT_VERSION_MINOR}" )

find_package( Qt6 COMPONENTS ShaderTools Widgets Qml Quick QuickControls2 LinguistTools REQUIRED )
qt_standard_project_setup( REQUIRES 6.5 )

add_subdirectory( src/flib )
add_subdirectory( src/flib_ui )
add_subdirectory( src/app )
```

Рис. 32 Приклад скрипту додатку на CMake

В даному прикладі, «src/flib» - є коміркою для бібліотеки ForwardAPI яка надає моєливість комунікації з сервером. «src/app» виступає самим UI додатком. На рис. 33 зображено дочірній опис самої програми з інтерфейсом, яка потім буде зібрана і відправлена в експлуатацію.

```
qt_policy( SET QTP0001 NEW ) # Ensures that modules are put into the QML Import Path and can be found without f
qt_policy( SET QTP0002 NEW ) # Support for cmake generator expressions in android json builds
qt_policy( SET QTP0004 NEW ) #

set( APP_TARGET "flib.app" )
set( APP_INCLUDE_DIR "${CMAKE_CURRENT_SOURCE_DIR}/include" )

qt_add_executable( ${APP_TARGET} ${SOURCES} MANUAL_FINALIZATION )

target_compile_definitions( ${APP_TARGET}
    PUBLIC APP_NAME="${FORWARD_APP_NAME}" APP_VERSION="${FORWARD_APP_VERSION}" APP_DOMAIN="${FORWARD_DOMAIN}" )

target_link_libraries( ${APP_TARGET}
    PRIVATE Qt6::Widgets Qt6::Qml Qt6::Quick Qt6::QuickControls2 flib.ui flib.uiplugin )

target_include_directories( ${APP_TARGET}
    PUBLIC "$<BUILD_INTERFACE:${APP_INCLUDE_DIR}>" )

set_target_properties( ${APP_TARGET} PROPERTIES
    WIN32_EXECUTABLE TRUE
    MACOSX_BUNDLE TRUE
    QT_ANDROID_APP_NAME "${FORWARD_APP_NAME}"
    QT_ANDROID_VERSION_CODE "${PROJECT_VERSION_MAJOR}"
    QT_ANDROID_VERSION_NAME "${FORWARD_APP_VERSION}"
    QT_ANDROID_PACKAGE_NAME "${FORWARD_URI}.${APP_TARGET}"
    QT_ANDROID_PACKAGE_SOURCE_DIR "${PROJECT_SOURCE_DIR}/src/android"
)

# Register qtquickcontrols2.conf
qt_add_resources( ${APP_TARGET} "app_configs"
    BASE "config"
    FILES ${CONFIGS}
)

# Add images
qt_add_resources( ${APP_TARGET} "app_images"
    BASE "assets"
    FILES ${ASSETS_IMAGES}
)

# Shaders
qt_add_shaders( ${APP_TARGET} "app_shaders"
    BASE "assets"
    FILES ${ASSETS_SHADERS}
)
```

Рис. 33 Опис дочірнього UI додатку в проєкті.

13.3 Розробка бібліотеки ForwardAPI

REST API - це архітектурний стиль для розробки веб-сервісів, який дозволяє комунікацію та обмін даними між різними комп'ютерними системами за допомогою інтернет-протоколу HTTP.

Qt Meta-Object Language або QML - це декларативна мова програмування, що використовується для опису інтерфейсів користувача в фреймворку Qt [3].

SSL (Secure Sockets Layer) - це криптографічний протокол, який забезпечує захищену передачу даних по мережі, зокрема через інтернет. SSL використовується для шифрування даних, що передаються між клієнтом

(наприклад, веб-браузером) та сервером, тим самим забезпечуючи конфіденційність та цілісність даних.

SSL handshake (Secure Sockets Layer handshake) - це процес встановлення захищеного з'єднання між клієнтом і сервером за допомогою протоколу SSL або його нащадка TLS (Transport Layer Security). Цей процес відбувається перед тим, як розпочнеться фактичний обмін даними між клієнтом і сервером [7].

Було розроблено набір API та бібліотеки під назвою ForwardAPI з метою спрощення розробки додатків з використанням асинхронної моделі програмування. Цей набір API надають зручний інструментарій для роботи з різними аспектами розробки, включаючи мережеву комунікацію, інтерфейси користувача та бази даних.

ForwardAPI має набір методів для реалізації REST API, що дозволяє зручно створювати та керувати веб-сервісами за допомогою архітектурного стилю REST. Це забезпечує ефективну комунікацію та обмін даними між різними комп'ютерними системами.

Також було розроблено компоненти для роботи з інтерфейсами користувача, використовуючи Qt Meta-Object Language (QML) [3]. Це дозволяє програмістам описувати інтерфейси користувача декларативно та забезпечує швидку розробку кросплатформових додатків.

13.3.1 Розробка API бази даних

Одним з прикладів див. рис. 34 є інтерфейс конектора MySQL який дозволяє створювати підключення до баз даних MySQL асинхронно [12] та має зрозумілий набір методів для розробників що спрощує розробку інших додатків. Це дозволяє розробникам легко отримувати доступ до баз даних та веб-файлів і використовувати їх в своїх додатках.

```
class Database final
{
    sql::mysql::MySQL_Driver* driver_;
    Scope<sql::Connection> connection_;
    bool is_scheme;

private:
    static std::unordered_map<std::string, Ref<Database>> databases_;

    Database(sql::mysql::MySQL_Driver* driver);

public:
    ~Database();

    static Ref<Database> Innit(std::string_view db_name);
    static Ref<Database> Get(std::string_view db_name);
    static void Remove(std::string_view db_name);

    static bool HasDatabase(std::string_view db_name);

    bool Connect(sql::ConnectOptionsMap options);
    bool Connect(sql::ConnectOptionsMap options, std::exception& ec);

    bool Connect(std::string_view host, std::string_view user, std::string_view password);
    bool Connect(std::string_view host, std::string_view user, std::string_view password, std::exception& ec);

    void Close() const;

    void SetActiveSchema(std::string_view scheme);

    Ref<sql::ResultSet> Execute(std::string_view query) const;

    template<typename ...Args>
    Ref<sql::ResultSet> PrepareQuery(std::string_view query, Args&&... args)···

    bool IsConnected() const;

    Database(const Database&) = delete;
    Database& operator=(const Database&) = delete;

private:
    template<typename T>
    void BindValue(Ref<sql::PreparedStatement> const& statement, int index, T&& value)···

    void BindValueImpl(Ref<sql::PreparedStatement> const& statement, int index, bool value);
    void BindValueImpl(Ref<sql::PreparedStatement> const& statement, int index, int32_t value);
    void BindValueImpl(Ref<sql::PreparedStatement> const& statement, int index, uint32_t value);
    void BindValueImpl(Ref<sql::PreparedStatement> const& statement, int index, int64_t value);
    void BindValueImpl(Ref<sql::PreparedStatement> const& statement, int index, uint64_t value);
    void BindValueImpl(Ref<sql::PreparedStatement> const& statement, int index, double value);
    void BindValueImpl(Ref<sql::PreparedStatement> const& statement, int index, const char* value);
    void BindValueImpl(Ref<sql::PreparedStatement> const& statement, int index, std::string_view value);
    void BindValueImpl(Ref<sql::PreparedStatement> const& statement, int index, DateTime const& date);
};
```

Рис. 34 Реалізація MySQL інтерфейсу бази даних

13.3.2 Розробка API безпечного серверу

Для забезпечення безпеки комунікації, ForwardAPI використовує SSL протокол для шифрування даних [6] та забезпечення конфіденційності та цілісності. Було реалізовано клас див. рис. 35 для захищеного сервера, які використовують SSL handshake для безпечного встановлення з'єднання між клієнтом і сервером.

```
class SslServer
{
protected:
    ssl::context secure_context_;

private:
    size_t io_count_;
    net::io_context io_context_;
    std::vector<std::thread> io_sessions_;

    Ref<AsyncListener> listener_;

public:
    /**
     *  @param method   sets encription method of SSL connection.
     *                  by default it will use latest secure method
     *  @param io_count specify amount of threads to io
     */
    SslServer(ssl::context::method method, uint8_t io_count);

    void Listen(Ref<Endpoint const> const& endpoint);

    void SetupFileSslCert(std::string_view filename,
        net::ssl::context::file_format format = net::ssl::context::pem);
    void SetupFileSslCertKey(std::string_view filename, std::string_view pass = "",
        net::ssl::context::file_format format = net::ssl::context::pem);

    virtual void OnSocketAccept(beast::error_code ec, tcp::socket&& socket);
};
```

Рис. 35 Клас захищеного серверу

Даний клас використовується для розробки HTTP серверу [6]. Що є по суті інтерфесом HTTP сервера. Клас має свою універсальність, тому його можна використовувати не тільки в HTTP сервері, а й у всіх доступних сервер окрім UDP серверів.

13.3.3 Розробка API файлової системи серверу

Розроблюючи HTTP сервер було створено API файлової системи для веб файлів див. рис. 36. Це надасть можливість отримувати шлях та інформацію про веб файл, щоб потім відправити його як відповідь (response) до клієнта серверу.

Система веб файлів слугує як фільтр та надає можливість вказати директорію вебсайту. Для того щоб вказати директорію потрібно при ініціалізації класу «WebFileSystem» який зображено на рис. 36, передати першим параметром шлях до директорії у конструктор класу.

```
struct WebFileMeta
{
    std::string Name; MimeType Ext;
    std::string Relative;
    std::filesystem::path Base;

    // returns file name
    std::string File(bool extension = true) const {···
    // returns path to a file
    std::string FullPath() const {···

    // returns target file name
    std::string TargetName(bool extension = true) const {···
    // returns relative path and file
    std::string TargetPath(bool extension = true) const {···

    bool operator==(WebFileMeta const& right) const ···
};

class WebFilesSystem
{
    // web documents directory
    std::string web_root_;
    // all stored files in web site
    std::vector<WebFileMeta> files_;

public:
    /**
     *  @param web_root is a directory where web documents stored
     */
    WebFilesSystem(std::string_view web_root);

    void SetWebRoot(std::string_view web_root);
    std::string_view GetWebRoot() const;

    // search file by name
    std::optional<WebFileMeta> FindByTargetName(std::string_view target, bool extension) const;
    // search file by target path
    std::optional<WebFileMeta> FindByTargetPath(std::string_view target, bool extension = true) const;

    bool IsTargetNameExist(std::string_view target, bool extension) const;
    bool IsTargetPathExist(std::string_view target, bool extension) const;
};
```

Рис. 36 Клас системи веб файлів

Після цього, об’єкт класу «WebFileSystem» просканує файлову систему директорії яка була вказана розробником, для того щоб фільтрувати запити користувачів (клієнтів) серверу.

13.3.4 Розробка API шляхів серверу

Одним із ключових компонентів API HTTP серверів є рутинг [13] або точніше реєстрація шляхів у вебсайту. Користувач робить запит на HTTP сервер, а система у відповідь перевіряє чи зареєстрований даний шлях, чи ні. Тому було розроблено систему яка надає можливість розробнику легко зареєструвати шлях на сервері та перевірити його валідність. Вона була реалізована на фільтрі файлової системи «WebFileSystem» див. рис. 36.

```
class HttpRouter
{
private:
    //
    std::string index_;
    // default extension for registered routes
    MimeType def_ext_;

    // file system to store every document info
    Scope<WebFilesSystem const> wfs_;
    // folders where assets, styles, scripts stored
    std::set<std::string> content_folder_;
    // key: reg route, value: prepared route
    std::map<std::string, std::string> routes_;

public:
    HttpRouter(std::string_view web_root, std::string_view index = "index", std::string_view ext = "html");

    std::string GetPreparedTarget(std::string_view name) const;

    std::string GetRoutePath(std::string_view target) const;
    std::string GetContentPath(std::string_view content) const;

    void RegisterRoute(std::string_view target);
    void RegisterContent(std::string_view folder_name);

    bool IsTarget(std::string_view target) const;
    bool IsContent(std::string_view content) const;

    static bool IsTargetLegal(std::string_view target);

private:
    bool IsTargetIndex(std::string_view target) const;

    // checks if a target is valid
    // returns prepared target or index
    std::string PrepareRouteName(std::string_view target) const;
};
```

Рис. 37 Система шляхів HTTP

Також реалізован клас мета-шляхів «HttpRouter» див. рис. 37. Даний клас надає можливість відразу повернути повний шлях до файлу. На сам перед, цей клас дозволяє перевірити URL запит на валідність, чи не має шкідливих для системи запитів або символів. Він також перевіряє чи даний запит є посиланням на сайт чи посиланням на контент сайту, що допомагає відрізнити сторінку сайту від його вложення (картинок).

В цілому, ForwardAPI надає потужний набір інструментів та API для розробки різноманітних додатків, забезпечуючи швидку і ефективну розробку та спрощуючи багато аспектів програмування, таких як мережева комунікація, інтерфейс користувача та робота з базами даних.

13.4 Розробка вебсайту

У межах реалізації програмного забезпечення месенджеру було створено допоміжний вебсайт, який виконує обмежені, але важливі функції — завантаження клієнтських застосунків та реєстрація нових користувачів.

Основне призначення вебсайту полягає в забезпеченні зручного доступу до платформи для нових користувачів, які можуть швидко ознайомитися з продуктом, обрати версію для своєї операційної системи (Windows, Linux або Android) та пройти реєстрацію в системі без потреби попередньої інсталяції.

Інтерфейс вебсайту реалізовано за допомогою стандартних вебтехнологій — HTML, CSS та JavaScript [13] [8]. Дизайн орієнтований на простоту, швидкість завантаження та адаптивність до різних пристроїв. Вебсайт сумісний з усіма сучасними браузерами, що забезпечує його доступність навіть для користувачів зі слабкими системами або нестабільним інтернет-з'єднанням.

Форма реєстрації користувача передбачає введення мінімального набору даних: логін, пароль, електронна пошта або номер телефону. Після заповнення дані надсилаються через захищений HTTPS-запит на серверну частину системи, де створюється обліковий запис користувача.

У майбутньому можлива інтеграція додаткових функцій, зокрема онлайн-чату, сторінки підтримки або адміністративного кабінету для реєстрації підприємств.

Таким чином, вебсайт виступає в якості стартової точки для взаємодії з системою та виконує функції презентаційного і реєстраційного модуля, доповнюючи основну клієнт-серверну архітектуру месенджера.

13.4.1 Створення сторінок вебсайту

Для вебсайту месенджера було створено декілька сторінок які призначені для завантаження додатку, реєстрації та входу в обліковий запис.

Було розроблено головну сторінку, де можна дізнатись більше про месенджер, приклад зображено на рис. 38.

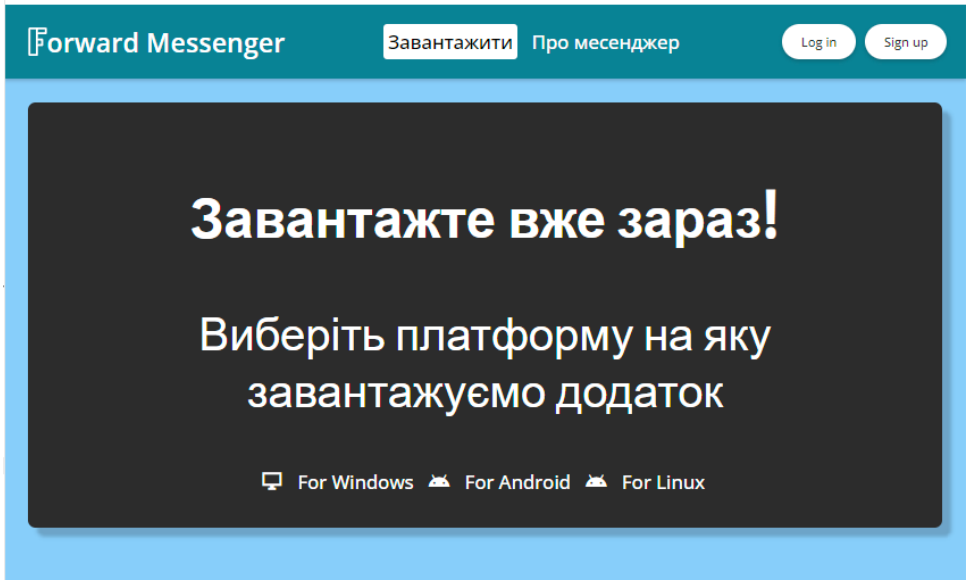


Рис. 38 Сторінка завантаження

13.4.2 Розробка лінивого завантаження

Ліниве завантаження (lazy loading) веб-ресурсів, таких як JavaScript і HTML, є підходом, який дозволяє завантажувати ресурси тільки тоді, коли вони потрібні або коли користувач досягає певного моменту на сторінці. Це може покращити швидкість завантаження сторінки та зменшити використання ресурсів.

В проєкті дана методика використовується при завантаженні картинок. Саме це допоможе завантажувати сайт швидко і з мінімальними витратами.

Intersection Observer – це API дає змогу відслідковувати, коли елемент на сторінці стає видимим в області браузера. Код на рис. 39 показує реалізацію динамічного завантаження ресурсів [14], коли вони стають видимими на сторінці.

```
const blurDivs = document.querySelectorAll(".load-animation")

blurDivs.forEach(div => {

    const img = div.querySelector("img");

    function loaded() {
        div.classList.add("loaded")
    }

    if (img.complete) {
        loaded()
    }
    else {
        img.addEventListener("load", loaded)
    }
})
```

Рис. 39 Код линивого завантаження

Також було створено декілька елементів стилю для відображення картинок, як показано на рис. 40.

```
.load-animation::before {
    content: "";
    position: absolute;
    inset: 0;
    animation: pulse 2.5s infinite;
}

.load-animation.loaded::before {
    content: none;
}

@keyframes pulse {
    0% {
        background-color: rgba(0, 0, 0, 0);
    }
    50% {
        background-color: rgba(0, 0, 0, 0.3);
    }
    100% {
        background-color: rgba(0, 0, 0, 0);
    }
}
```

Рис. 40 Стиль лінивого завантаження

Реалізована анімація відтворює пульсацію що напам’ятовує завантаження. Також було додано картинки низкої якості для швидшого відтворення їх.

13.5 Розробка серверної частини

Для розробки серверної частини було використано такі методи, як: бібліотеки Boost.Asio [15], Boost.Beast [16] та розроблена ForwardAPI.

Завдяки даному набору технологій можна створювати REST сервера, що дасть легку комуніцацію між клієнтами та сервером.

13.5.1 Розробка HTTPS серверу

Розроблено сервер Рис. 41 за допомогою ForwardAPI, а саме SslServer.

Даний код представляє клас HttpServer, який успадковується від класу SslServer і використовує бібліотеки Boost.Asio та Boost.Beast для розробки серверної частини. Цей клас дозволяє створювати HTTPS сервери та має наступний функціонал:

- router_ і request_handler_ - приватні поля, що представляють маршрутизатор та обробник запитів HTTP.
- user_sessions_ - приватне поле, що зберігає список активних сесій користувачів.
- Конструктор HttpServer приймає шлях до директорії веб-ресурсів, метод шифрування SSL та кількість потоків введення/виведення (io_count) для Boost.Asio.

Метод SetContentFolders приймає вектор рядків, що представляють шляхи до додаткових папок з вмістом. Ці папки будуть використовуватись для обробки запитів на статичні ресурси.

Метод SetBadRequest приймає функцію-обробник (BadRequest), яка викликається при отриманні некоректного запиту.

```
class HttpServer final : public SslServer
{
private:
    Ref<HttpRouter> router_;
    Ref<HttpRequestHandler> request_handler_;

    std::list<Ref<HttpSession>> user_sessions_;

public:
    HttpServer(std::string_view web_dir, net::ssl::context::method method, uint8_t io_count);

    void SetContentFolders(std::vector<std::string> const& folders);
    void SetBadRequest(HttpRequestHandler::BadRequest const& handler);

    template<typename ...Args>
    void Route(Args&& ...args)
    {
        HttpRequestHandler::Data data {std::forward<Args>(args)...};

        if (!data.Target.empty())
            router_->RegisterRoute(data.Target);

        if (data.Handler.has_value() || data.Method.has_value())
            request_handler_->AddRouteHandler(data);
    }

    void OnSocketAccept(beast::error_code ec, tcp::socket&& socket) override;
};
```

Рис. 41 Реалізація HTTPS серверу

Шаблонний метод Route дозволяє реєструвати маршрути та обробники запитів. Він приймає декілька аргументів, які передаються у конструктор HttpRequestHandler::Data. Якщо Target не є порожнім рядком, маршрут реєструється у маршрутизаторі. Якщо Handler або Method мають значення, обробник запиту додається до обробника запитів.

Метод OnSocketAccept є віртуальним методом, успадкованим від SslServer. Він викликається при прийнятті нового з'єднання на сервері.

Цей клас дозволяє створювати універсальні HTTPS сервери з підтримкою шифрування SSL, реєстрацією маршрутів запитів, обробкою подій, відправкою файлів та текстових відповідей. Він забезпечує зручний інтерфейс для розробки серверної частини додатків, що працюють з HTTP протоколом.Даний клас є універсальний сервером, який може надати шифрування завдяки SSL, реєстрацію URL запитів, додавати події зареєстрованій сторінці, відправляти файли та текст.

13.5.2 Розробка класу сесій

Для підключення клієнтів до серверу, було розроблено клас «HttpSession» [17] див. рис. 42. Даний клас використовує бібліотеки Beast та Asio [15] для роботи з HTTP. Сесія також використовує обробник запитів.

Клас HttpSession має такі поля як: api_ який представляє обробник запитів HTTP; stream_ що представляє SSL-зашифроване з'єднання; buffer_ використовується для зберігання буфера даних, отриманих від клієнта; req_ представляє HTTP запит від клієнта.

Метод Run() запускає сесію HTTPS. Він викликає внутрішній приватний метод OnRun(), який розпочинає рукостискання та потім обробку запитів та відповідей.

```
class HttpSession : public std::enable_shared_from_this<HttpSession>
{
private:
    Ref<HttpRequestHandler> api_;

    beast::ssl_stream<beast::tcp_stream> stream_;
    beast::flat_buffer buffer_;

    http::request<http::string_body> req_;

public:
    // Take ownership of the socket
    explicit HttpSession(
        Ref<HttpRequestHandler> const& api,
        tcp::socket&& socket,
        net::ssl::context& context
    );

    // Start the session
    void Run();

private:
    void OnRun();

    void OnHandshake(beast::error_code ec);

    void DoRead();
    void OnRead(beast::error_code ec, std::size_t bytes_transferred);

    // Called to start/continue the write-loop. Should not be called when
    // write_loop is already active.
    void DoWrite(http::message_generator&& msg);
```

Рис. 42 Реалізація класу сесій

Цей клас використовується для обробки окремої HTTPS сесії, зчитування запитів від клієнта, відправлення відповідей та керування циклом сесії.

13.5.3 Встановлення та підключення серверу

Щоб встановити сервер, перш за все потрібно налаштувати його. З файлу config/server.json будуть завантажені дані: адреса підключеня (зазвичай вона локальна), сертифікат та ключ SSL і вказана директорія веб проєкту.

```
auto server_json = json::parse(std::move(json_doc), json_ec).as_object();

if (json_ec)
{
    FL_LOG("Parser", "Can not parse config/server.json");
    return EXIT_FAILURE;
}

auto cert = server_json["cert"].as_object();

auto cert_file =  cert["cert-file"].as_string();
auto cert_key =  cert["cert-key"].as_string();
auto cert_phrase =  cert["cert-phrase"].as_string();

auto web_dir = server_json["web_dir"].as_string();
```

Рис. 43 Завантаження даних з json

На рис. 43 зображено об’єкт json який потім конвертується в поля які потім використовуються сервером.

Потім створюємо об’єкт сервер як на рис. 44, передаємо шлях до веб сторінки (локально), вказуємо метод шифрування, кількість потоків та сертифікат шифрування.

```
HttpServer server(web_dir, net::ssl::context::tlsv13, 8);

server.SetupFileSslCert(cert_file);
server.SetupFileSslCertKey(cert_key, cert_phrase);

server.SetContentFolders({"assets", "style", "scripts"});
```

Рис. 44 Вказівка змінних в сервер

Із json об’єкту, повертаємо ip та порт серверу і потім встановлюєм прослуховування адреси як на рис. 45.

```
auto server_ip = server_json["ip"].as_string();
auto server_port = server_json["port"].as_string();

auto args = StringArg::MakeArgs({
    {"ip", server_ip.c_str()},
    { "port", server_port.c_str() }
});
StringBuilder server_adress("ip:port", args);

server.Listen(MakeRef<Endpoint>(server_adress.Data()));
```

Рис. 45 Дані для прослуховування адресу

13.5.4 Реєстрація запитів URL

Перед підключенням сервера, потрібно зареєструвати шляхи на які користувач може звертатись за веб сторінкой чи статичним файлом. Також

можна реалізувати за допомогою цього класу REST API. Для реєстрації шляху потрібно викликати метод Route, приклад використання зображено на рис. 46.

```
server.Route("/create", http::verb::get);
server.Route("/create", http::verb::post,
[&db](HttpRequest const& req, HttpResponse&& res) {

    auto userData = req.Query();

    FL_LOG("Action", "create");

    auto username(userData.Value("username"));
    auto email(userData.Value("email"));
    auto password(userData.Value("password"));

    return res;
});

server.Route("/login", http::verb::get);
server.Route("/login", http::verb::post,
[&db](HttpRequest const& req, HttpResponse&& res) {

    auto userData = req.Query();

    FL_LOG("Action", "login");

    auto email(userData.Value("email"));
    auto password(userData.Value("password"));

    return res;
});
```

Рис. 46 Реєстрація шляхів

Якщо користувач або зловмистник зробить запит в не зареєстрований шлях, то на такий випадок реалізовано метод див. рис. 47 який виконує реалізований метод розробника.

```
server.SetBadRequest(
[](HttpRequest const& req, http::status status)
{
    HttpResponse res{ status, req.Version() };
    res.SetHeader(http::field::server, BOOST_BEAST_VERSION_STRING);
    res.SetHeader(http::field::content_type, "text/html");
    res.SetAlive(req.Alive());

    switch (status)
    {
    case http::status::not_found: res.Body() = "The resource was not foun
        break;
    case http::status::bad_request: res.Body() = "Bad request";
        break;
    case http::status::internal_server_error: res.Body() = "Internal serv
        break;
    case http::status::unknown: res.Body() = "Unknown";
        break;
    }

    res.Prepare();
    return res;
});
```

Рис. 47 Реалізація некоректного запиту

13.5.5 Підключення бази даних

Для підключення бази даних, було використано клас з бібліотеки ForwardAPI який реалізовує простий інтерфейс для підключення баз даних див. рис. 48 підключення бази даних. При підключенні до бази даних строка підключення завантажується з файлу JSON в об’єкт класу JSON. Після цього, завантажені дані ми передаємо в рядок підключення бази даних до серверу MySQL [18].

```
auto db = Database::Innit("forward-db");
auto db_schema = db_json["active-schema"].as_string();

while (!db->IsConnected())
{
    std::exception db_ec;

    auto db_ip = db_json["ip"].as_string();
    auto db_port = db_json["port"].as_string();

    auto db_user = db_json["user"].as_string();
    auto db_pass = db_json["password"].as_string();

    auto args = StringArg::MakeArgs({
        {"ip", db_ip.c_str()},
        { "port", db_port.c_str() }
    });
    StringBuilder db_adress("tcp://ip:port", args);

    db->Connect(db_adress.Data(), db_user, db_pass, db_ec);

    if (!db->IsConnected())
        FL_LOG("Database", db_ec.what());
}

db->SetActiveSchema(db_schema);

FL_LOG("Database", "Connected");
```

Рис. 48 Підключення бази даних

Розроблений інтерфейс бази даних виконує запити асинхроно [19], це дозволяє уникнути затримки при виконанні коду, що робить код продуктивним та швидким.

13.5.6 Налаштування серерверу через JSON

JavaScript Object Notation або JSON - це формат обміну даними, який використовується для передачі структурованої інформації між різними системами. Він базується на синтаксисі мови JavaScript, але є незалежним від конкретної мови програмування і може використовуватись у багатьох інших мовах.

Щоб сервер був універсальним у підключеннях, було розроблено структуру JavaScript Object Notation, яка дозволяє зберігати данні у форматі який з легкістю читається розробником. Веб сервер має декілька конфігураційних файлів для стабільної роботи. Конфігурація для бази даних зображена на рис. 49.

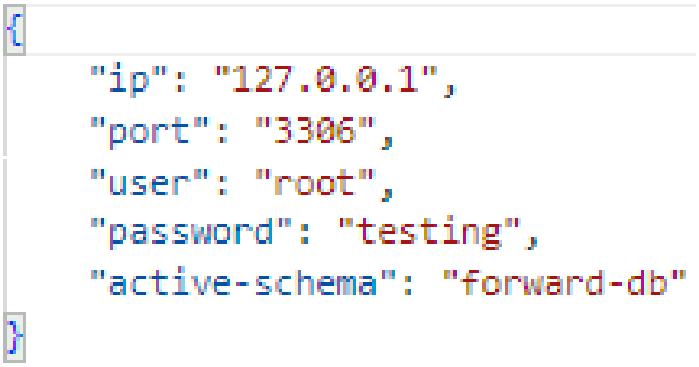


Рис. 49 Конфігурація бази даних

Структуру конфігурації для веб серверної частини зображено на рис. 50.

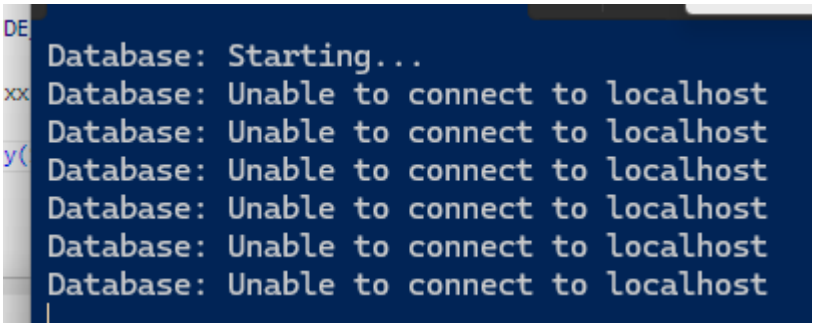
Рис. 50 Конфігурація веб серверу

Кожна конфігурація зчитується веб сервером та зчитані дані використовуються у сервері.

13.5.7 Запуск веб серверу

Щоб запустити веб сервер, по перше, потрібно вказати конфігураційні файли. В випадку не правильних даних, веб сервер оповістить що вказані данні були введені не вірно як показано на рис. 51. Перед завантаженням потрібно вказати шлях до наявнявного вебсайту в конфігурацію див. рис. 50, щоб веб сервер його хостив.

Сервер також перевіряє дійсність конфігурації, тому у випадку якщо данні конфігурації були невірно введені, у терміналі з’явиться повідомлення про помилку як на рис. 51.



```
Database: Starting...
Database: Unable to connect to localhost
Database: Unable to connect to localhost
Database: Unable to connect to localhost
Database: Unable to connect to localhost
Database: Unable to connect to localhost
Database: Unable to connect to localhost
```

Рис. 51 Повідомлення про «не правильне підключення»

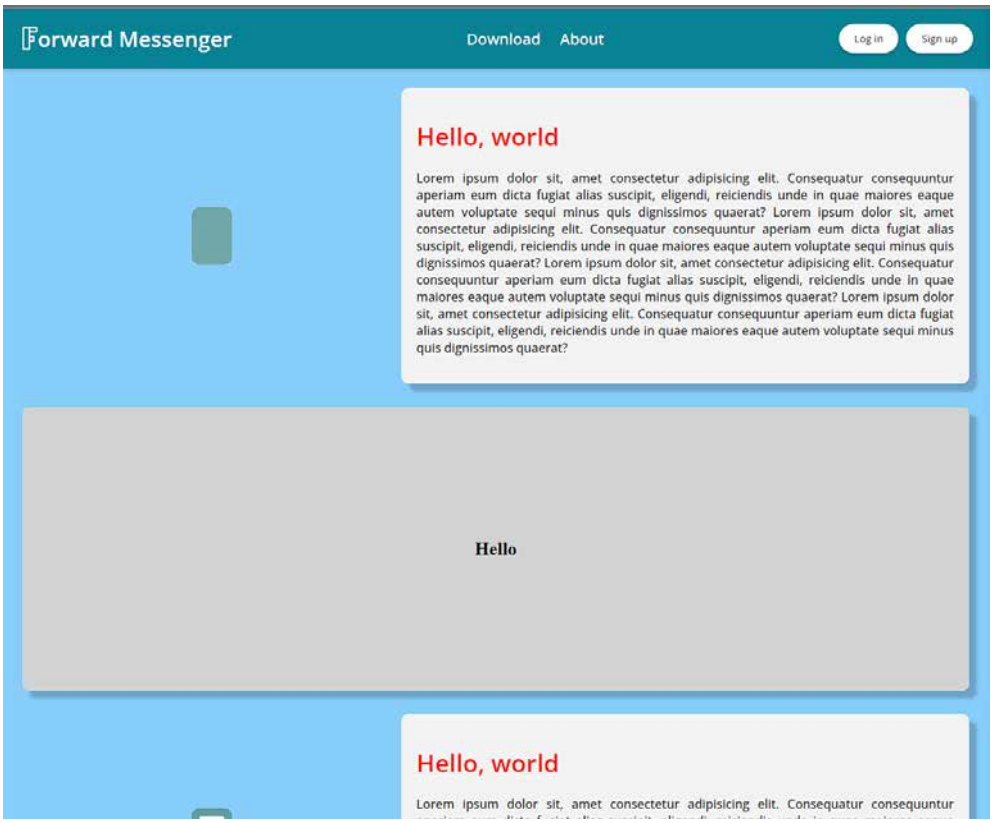
У випадку якщо сервер підключено правильно, у терміналі буде зображено повідомлення про успішне підключення як на рис. 52.

```
Database: Connected
WebServer: Starting...
SslServer: Started at 0.0.0.0:8080
```

Рис. 52 Повідомлення про «успішне пікдлючення»

Після запуску веб сервера, відкриваємо вікно браузеру та вводимо у пошукове поле адресу яка була вказана в конфігураціїї серверу див. рис. 50.

Браузер повинен завантажити сторінку яку було підключено до серверу, приклад завантаженої сторінки на рис. 53.



Hello, world

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Consequatur consequuntur aperiam eum dicta fugiat alias suscipit, eligendi, reiciendis unde in quae maiores eaque autem voluptate sequi minus quis dignissimos quaerat? Lorem ipsum dolor sit, amet consectetur adipisicing elit. Consequatur consequuntur aperiam eum dicta fugiat alias suscipit, eligendi, reiciendis unde in quae maiores eaque autem voluptate sequi minus quis dignissimos quaerat? Lorem ipsum dolor sit, amet consectetur adipisicing elit. Consequatur consequuntur aperiam eum dicta fugiat alias suscipit, eligendi, reiciendis unde in quae maiores eaque autem voluptate sequi minus quis dignissimos quaerat? Lorem ipsum dolor sit, amet consectetur adipisicing elit. Consequatur consequuntur aperiam eum dicta fugiat alias suscipit, eligendi, reiciendis unde in quae maiores eaque autem voluptate sequi minus quis dignissimos quaerat?

Hello

Hello, world

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Consequatur consequuntur

Рис. 53 Успішна робота вебсайту на сервері

13.5.8 Хостинг вебсайту

Хостинг проєкту здійснений за допомогою розробленого веб серверу. Даний сервер розроблений з використанням методів таких як C++ та засобів Boost.Beast та ForwardAPI. Сервер високопродуктивний [19] тому можна розроблювати сайти не хвилюясь за завантаження сторінок. На рис. 54 зображено приклад запущеного сайту з сервереру реалізованого на C++.

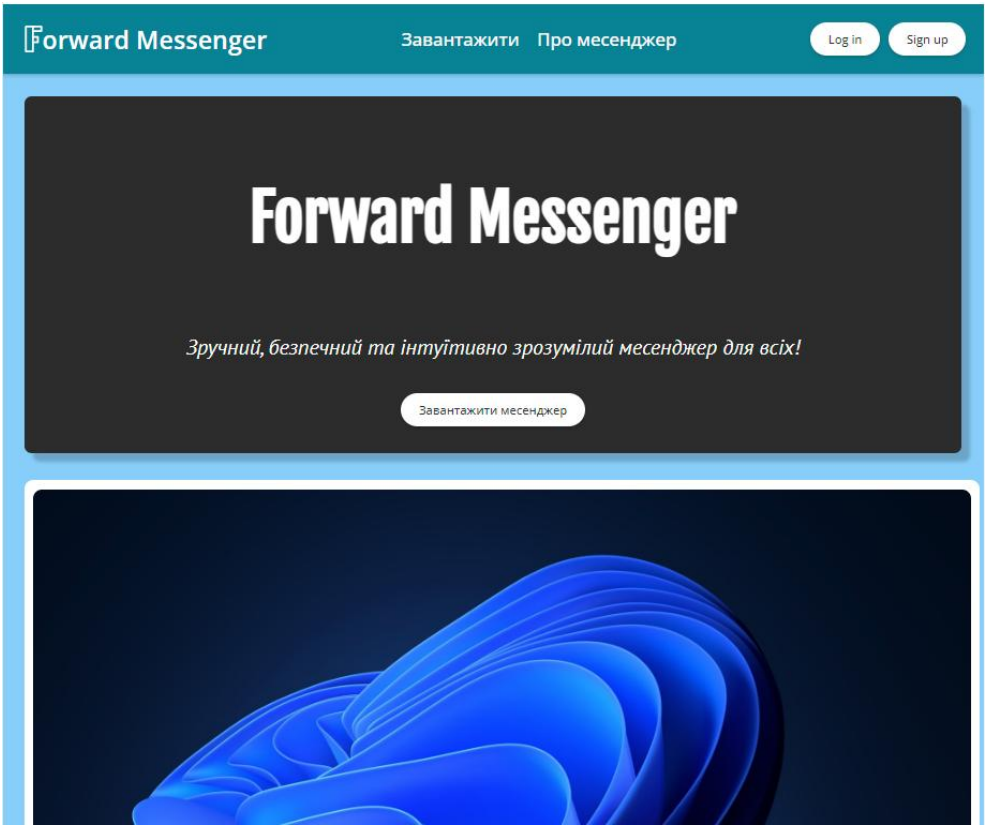


Рис. 54 Робота веб серверу з вебсайтом

До серверу також було налаштовано домен forchat.online. А сам сервер запущено на віртуальній машині.

13.6 Розробка клієнтського додатку

Щоб розробити кросплатформенний додаток було використано Qt Framework, QML [3] [20], а також розроблено бібліотеки для графічного інтерфейсу які не підтримує Qt Framework.

13.6.1 Розробка бібліотеки графічного інтерфейсу

QML сам по собі не підтримує деякі елементи UI, тому було розроблено графічну бібліотеку UI елементів. Дана бібліотка використвовує компоненти Qt Framework, а саме застарілі UI елементи та перетворює на новітню версію для QML.

Перш за все, потрібно створити проєкт бібліотеки графічного інтерфейсу, тому було використано генератор сценарії CMake як на рис. 55.

```
set( INCLUDES

    include/CursorPosition.hpp
    include/SystemNotifier.hpp

    include/ChatItemModel.hpp
    include/MessageItemModel.hpp
)

set( SOURCES
    ${INCLUDES}

    src/CursorPosition.cpp
    src/SystemNotifier.cpp

    src/ChatItemModel.cpp
    src/MessageItemModel.cpp
)

qt_policy( SET QTP0001 NEW ) # Ensures that modules are put into the QML Import
qt_policy( SET QTP0004 NEW ) #

set( FLUI_TARGET "flib.ui" )
set( FLUI_INCLUDE_DIR "${CMAKE_CURRENT_SOURCE_DIR}/include" )

qt_add_library( ${FLUI_TARGET} STATIC MANUAL_FINALIZATION )

target_link_libraries( ${FLUI_TARGET} PRIVATE Qt6::Widgets Qt6::Qml Qt6::Quick )
target_include_directories( ${FLUI_TARGET}
    PUBLIC "$<BUILD_INTERFACE:${FLUI_INCLUDE_DIR}>" )

qt_add_qml_module( ${FLUI_TARGET}
    URI "${FLUI_TARGET}"
    VERSION 1.0
    SOURCES ${SOURCES}

    # QML_FILES ${QML_SOURCES}
    OUTPUT_DIRECTORY "flib/ui"
    PLUGIN_TARGET "${FLUI_TARGET}plugin"
    RESOURCE_PREFIX "${FORWARD_URI}/imports"
)

qt_finalize_target( ${FLUI_TARGET} )
```

Рис. 55 Код створення проекту бібліотеки

На приклад, візьмемо клас системного трею який не підтримується QML, але його можна реалізувати за допомогою C++ та імпортувати саме в QML [20]. Приклад даного класу зображено на рис. 56

```
class SystemNotifier : public QObject
{
    Q_OBJECT
    QML_ELEMENT

    Q_PROPERTY(QUrl trayIcon READ trayIcon WRITE setTrayIcon);
    Q_PROPERTY(QUrl messageIcon READ messageIcon WRITE setMessageIcon);

    Q_PROPERTY(bool visible READ isVisible WRITE setVisible);

public:
    SystemNotifier(QObject *parent = Q_NULLPTR);
    ~SystemNotifier() override;

    void setTrayIcon(const QUrl &source);
    void setMessageIcon(const QUrl &source);

    void setVisible(const bool visible);

    inline bool isVisible() const { return m_SystemTray->isVisible(); }

    inline QUrl trayIcon() const { return m_TrayIcon; }
    inline QUrl messageIcon() const { return m_MessageIcon; }

    inline QMenu *menuContext() const { return m_SystemTray->contextMenu(); }

    Q_INVOKABLE void showMessage(
        const QString &title,
        const QString &msg,
        const QUrl &icon,
        int millisecondsTimeoutHint = 10000);

private slots:
    void onTrayActivated(const QSystemTrayIcon::ActivationReason reason);

signals:
    void triggered();
    void messageClicked();

private:
    QUrl m_TrayIcon;
    QUrl m_MessageIcon;
    std::unique_ptr<QSystemTrayIcon> m_SystemTray;
};
```

Рис. 56 Реалізація трею додатка

Клас «SystemTray» див. рис. 56 було реалізвано як обгортку на системі віджетів. Тому даний клас перезаписує методи та надає можливість використовувати їх у мові QML.

```
Client.SystemTray {
    id: appTray

    iconTray: ":/forward/logo_64.png"
    iconMsg: ":/forward/logo_64.png"

    menu: Client.TrayMenu {
        title: app.title
        icon: ":/forward/logo_64.png"

        Client.TrayMenuItem {
            text: qsTr("Sing out")

            onClicked: {}
        }
        Client.TrayMenuItem {
            text: qsTr("Quit Forward")

            onClicked: app.close()
        }
    }

    onMessageClicked: app.show()

    onTrayContext: menu.popup(Client.CursorPosition.pos())
    onTrayDoubleClicked: app.show()
}
```

Рис. 57 Використання класу

Після реалізацію «SystemTray» ми його сторюємо в мові QML, передаємо параметри ініціалізації та компілюємо проєкт. На рис. 58 показана робота скомпільованого класу QML який реалізує трей-меню в комп'ютерних системах.

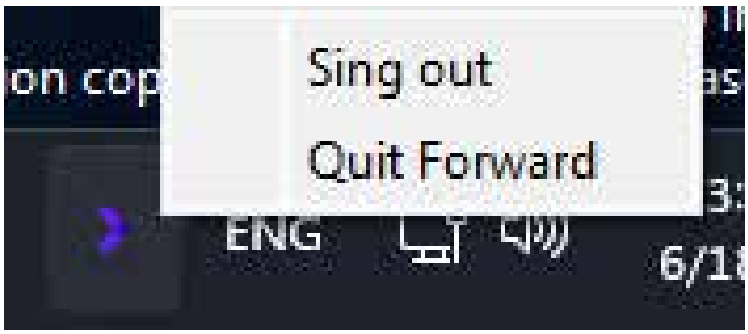


Рис. 58 Робота системного трею

Також даний клас надає можливість відправляти сповіщення у виведеному вікні операційної системи, як показано на рис. 59.

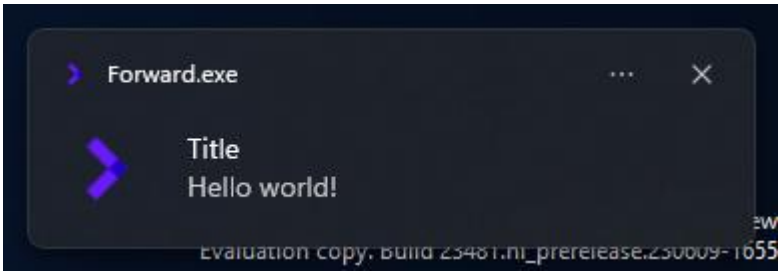


Рис. 59 Сповіщення в системі

Дане API дуже важливе для сповіщень коли додаток в згорнутому режимі, що дозволяє використовувати сповіщення від операційної системи. Щоб вікно працювало при натисканні, потрібно реалізувати одну обгортку класу тому що знову Qt Framework не підтримує дані елементи графічного інтерфейсу в мові розмітки QML.

```
class TrayMenuWrapper : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString title READ GetTitle WRITE SetTitle)
    Q_PROPERTY(QString icon READ GetIcon WRITE SetIcon)
    Q_PROPERTY(QList<QObject*> menuItems WRITE SetMenuItemList NOTIFY menuItemsChanged)
    QML_ELEMENT

private:
    QString icon_;
    QMenu* menu_;

public:
    explicit TrayMenuWrapper(QObject* parent = nullptr);
    ~TrayMenuWrapper() override;

    void SetTitle(const QString& title);
    void SetIcon(const QString& icon);
    void SetMenuItemList(const QList<QObject*>& list);

    QString GetTitle() const;
    QString GetIcon() const;

    Q_INVOKABLE void popup(const QPoint& point);

signals:
    void menuItemsChanged();

};
```

Рис. 60 Реалізація вікна з випадаючим списоком

Даний клас зображено на рис. 60, реалізовує вікно з випадаючим списком який легко налаштовується і працює без проблем. Потім його можна легко застосувати в мові розмітки QML, як зображено на рис. 61.

```
menu: Client.TrayMenu {
    title: app.title
    icon: ":/forward/logo_64.png"

    Client.TrayMenuItem {
        text: qsTr("Sing out")

        onClicked: {}
    }
    Client.TrayMenuItem {
        text: qsTr("Quit Forward")

        onClicked: app.close()
    }
}
```

Рис. 61 Використання класу в QML

13.6.2 Розробка API локалізації та переклад додатку

Щоб додатком користувались користувачі з інших країн було розроблено систему для зміни мови зображену на рис. 62.

```
class Localization : public QObject
{
    Q_OBJECT
    QML_SINGLETON
    QML_ELEMENT

private:
    static QTranslator* translator_;

public:
    Q_INVOKABLE QStringList localeList() const;
    Q_INVOKABLE QStringList countryList() const;

    Q_INVOKABLE QString currentLocale() const;

    Q_INVOKABLE void changeLocale(QString const& language);

    QString getLanguageCode(QString const& displayName) const;
};
```

Рис. 62 Клас для зміни мови

Даний клас має свої обмеження, але виконує поставлені задачі такі як зміна мови, отримання поточної мови системи, список країн, список підтримуючих трансляцій.

Кожен файл трансляції було реалізовано з використанням засобу Qt Linguist як зображено на рис. 63.

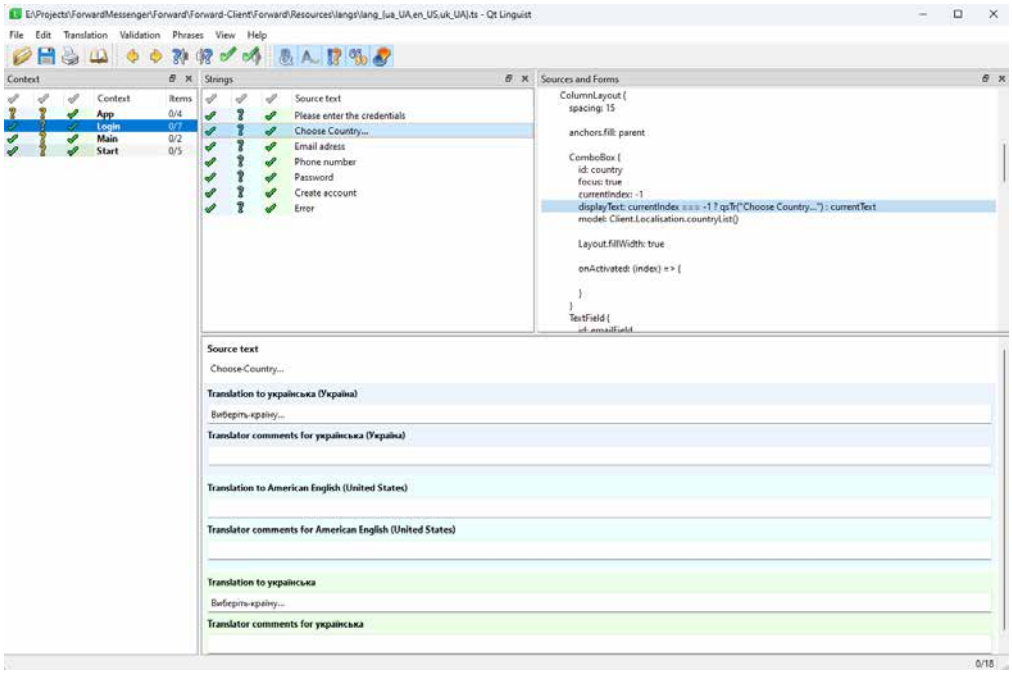


Рис. 63 Інтрумент Qt Linguist

Після написання файлів трансляції та API зміни тексту їх було додано до проєкту. Приклад використання класу для зміни мови в QML показано на Рис. 64.

```
Component.onCompleted: {
    const lang = appSettings.valueAt("lang", Client.Localization.currentLocale())
    const index = indexOfValue(lang)

    console.log("Current language: " + lang)

    currentIndex = index
}
onActivated: (index) => {
    const lang = textAt(index)

    appSettings.addValue("lang", lang)
    appSettings.save()

    Client.Localization.changeLocale(lang)
}
```

Рис. 64 Частина коду для зміни мови

13.6.3 Розробка API тем додатку

Для гарного вигляду додатка, було створено декілька класів які налоштовують загальний вигляд графічного інтерфейсу. Один з таких класів є

«Style», реалізацію зображено на рис. 65, який надає можливість зміни стилю додатку за ім’ям або подивитись доступні теми.

```
class Style : public QObject
{
    Q_OBJECT
    QML_SINGLETON
    QML_ELEMENT

public:
    Q_INVOKABLE QString currentStyle() const;
    Q_INVOKABLE QStringList supportedStyles() const;

    Q_INVOKABLE void changeStyle(QString const& name);
};
```

Рис. 65 Клас для зміни стилю додатка

У додатку при першому завантеженні, відкриється меню з мовою та темою. Користувач може вибрати декілька заготовлених тем як на рис. 66.

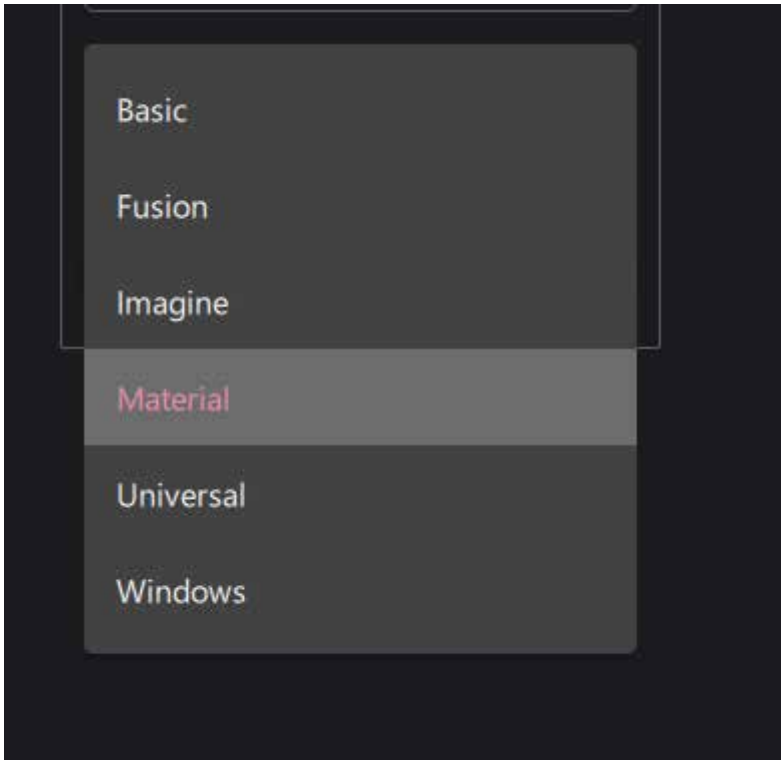


Рис. 66 Вибір теми додатку

Далі, додаток запропонує перезавантажити себе як на рис. 67. Після повторного відкритя мова буде змінена на обрану мову користувачем.

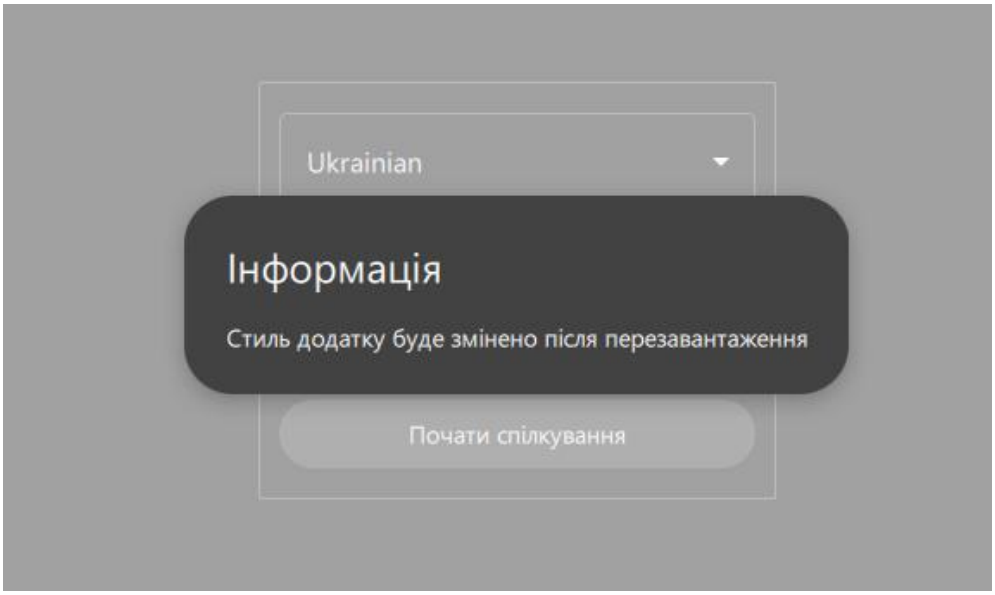


Рис. 67 Діалогове повідомлення про перезавантаження

Коли користувач обирає тему, обрана тема зберігається в налаштуваннях операційної системи.

13.6.4 Розробка API збережень налаштувань додатку

Для збереження даних які не потрібно створювати по декілька разів, було розроблено клас який надає можливості збереження даних. Клас реалізован на C++ та є плагіном для мови QML.

```
class SettingsWraper : public QObject
{
    Q_OBJECT
    QML_NAMED_ELEMENT(Settings)

private:
    QSettings* settings_;

public:
    explicit SettingsWraper(QObject* parent = nullptr);
    ~SettingsWraper() override;

    Q_INVOKABLE void save() const;

    Q_INVOKABLE void addValue(QString const& key, QVariant const& value);

    Q_INVOKABLE QVariant valueAt(QString const& key) const;
    Q_INVOKABLE QVariant valueAt(QString const& key, QVariant const& default_value) const;

    Q_INVOKABLE bool isValueAt(QString const& key) const;
};
```

Рис. 68 Розроблений клас налаштувань

Даний клас, зображено на рис. 68 реалізовує операції додавання та вибору збережених налашувань. Клас має бути включен при розробці в мові QML, тому також прописуємо його як на рис. 69.

```
Client.Settings {
    id: appSettings
}
```

Рис. 69 Реалізація налаштувань в мові QML

41 ТЕСТУВАННЯ МЕСЕНДЕЖЕРУ

14.1 Тестування бібліотеки ForardAPI

Google Test є ефективним способом перевірки функціональності програмного коду в мові C++. Google Test допомагає розробникам виявити та виправити помилки та неполадки ще до того, як програма буде випущена у продакшн. Google Test також підтримується генератором CMake, тому було обрано саме дане рішення.

Щоб CMake бачив тестувальні проєкти, написано скрипти див. рис. 70 під кожну частину бібліотеки.

```
cmake_minimum_required(VERSION 3.24)

set(TEST_SOURCE
    test_flags.cpp
    test_stringarg.cpp
    test_stringbuilder.cpp
    # test_log.cpp
    test_datetime.cpp
    test_mimetype.cpp
)

# For Windows: Prevent overriding the parent project
set(gtest_force_shared_crt ON CACHE BOOL "" FORCE)

enable_testing()

add_executable(TestUtils ${TEST_SOURCE})

target_link_libraries(TestUtils
PUBLIC
    GTest::gtest_main
    Fl::utils
)

include(GoogleTest)
gtest_discover_tests(TestUtils)
```

Рис. 70 Створений проєкт для тестування компоненту

Після відображення в системі CMake, створено декілька тестів утилітного компоненту «DateTime» див. рис. 71. Даний клас надає засоби для відображення та форматування часу за різними системами. Він часто використовується в базі даних та мережевої частинах, тому для нього були написані тестові сценарії які допоможуть тримати компоненти без багів.

```
#include "fl/utils/DateTime.hpp"
using namespace fl;

#include <gtest/gtest.h>

TEST(DateTime, ToCFormat)
{
    std::string out = DateTime::ConvertToCFormat("DD-MM-YYYY hh-mm-ss");

    EXPECT_STREQ(out.c_str(), "%d-%m-%Y %H-%M-%S");
}

TEST(DateTime, NowToString)
{
    std::string now = DateTime::Now().ToString("DD-MM-YYYY hh-mm-ss");
    std::string latenow = DateTime::Now().ToString("DD-MM-YYYY hh-mm-ss");

    EXPECT_STREQ(now.c_str(), latenow.c_str());
}

TEST(DateTime, CopyEq)
{
    auto now = DateTime::Now();
    auto copy = now;

    EXPECT_TRUE(now == copy);
}

#include <optional>

TEST(DateTime, Exception)
{
    std::string now = DateTime::Now().ToString("dafsd   4252  2s");
    std::string latenow = DateTime::Now().ToString("strange text");
}
```

Рис. 71 Створений тест для класу «DateTime»

Таким чином, використання бібліотеки googletest у комбінації з C++ дозволяє розробникам ефективно тестувати свій програмний код, забезпечуючи високу якість та надійність програмного продукту.

14.2 Тестування вебсайту за допомогою Chrome Lighthouse

Під час розробки вебсайту важливо не лише створити візуально привабливий та зручний інтерфейс, а й забезпечити високу продуктивність системи. Продуктивність безпосередньо впливає на якість користувацького досвіду, рівень задоволеності користувачів, а також загальний імідж цифрового продукту. Повільна або нестабільна робота сайту може спричинити відтік відвідувачів, зниження довіри до ресурс та втрату потенційних клієнтів.

Одним із критично важливих етапів забезпечення належної роботи є тестування продуктивності вебресурсу. Такий підхід дозволяє перевірити, як система поводиться за умов реального навантаження, виявити вузькі місця та гарантувати стабільну роботу під час пікових періодів використання.

До основних аспектів тестування продуктивності належать оцінка здатності сайту обробляти велику кількість одночасних запитів, аналіз швидкості завантаження сторінок, перевірка стабільності системи під час тривалого використання, а також вивчення її масштабованості — тобто здатності адаптуватися до зростання обсягів даних або кількості користувачів. Крім того, доцільним є моделювання сценаріїв, що максимально наближені до реального використання, аби переконатися у стабільності роботи системи під різними умовами.

Для автоматизованого аналізу вебсайтів активно застосовуються спеціалізовані інструменти, зокрема Chrome Lighthouse — інтегрований засіб, розроблений командою Google. Lighthouse дає змогу проводити всебічну перевірку продуктивності, доступності, якості коду та дотримання сучасних вебстандартів. Інструмент формує докладний звіт із показниками та рекомендаціями, які дозволяють розробнику оперативно виявити й усунути проблеми, що впливають на швидкодію сайту та зручність його використання.

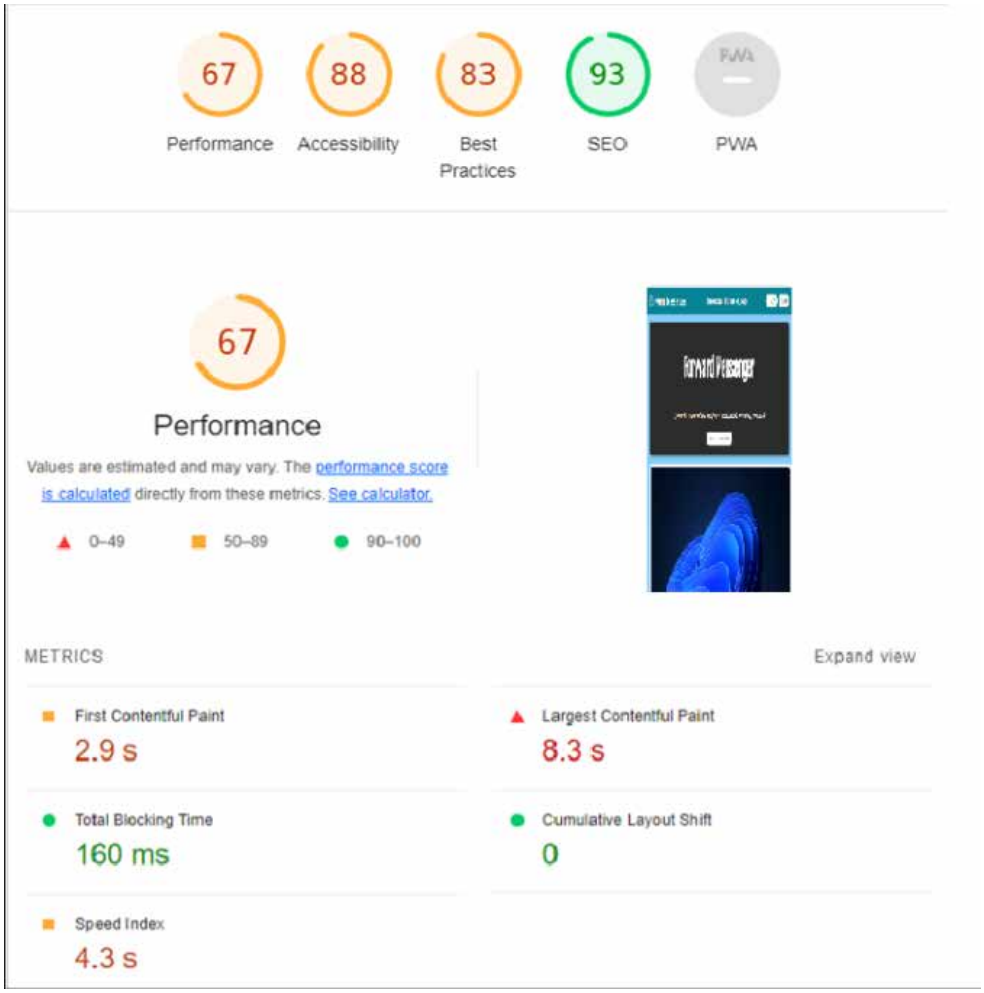


Рис. 72 Результат тестування вебсайту

Отже було виконано декілька тестувань вебсайту за допомогою утиліти Chrome LightHouse. На рис. 72 можна побачити що завантаження сайту в межах норми сайтів. Наприклад візьмемо сайт telegram, де завантаження відбувається швидше, як зображено на рис. 73.

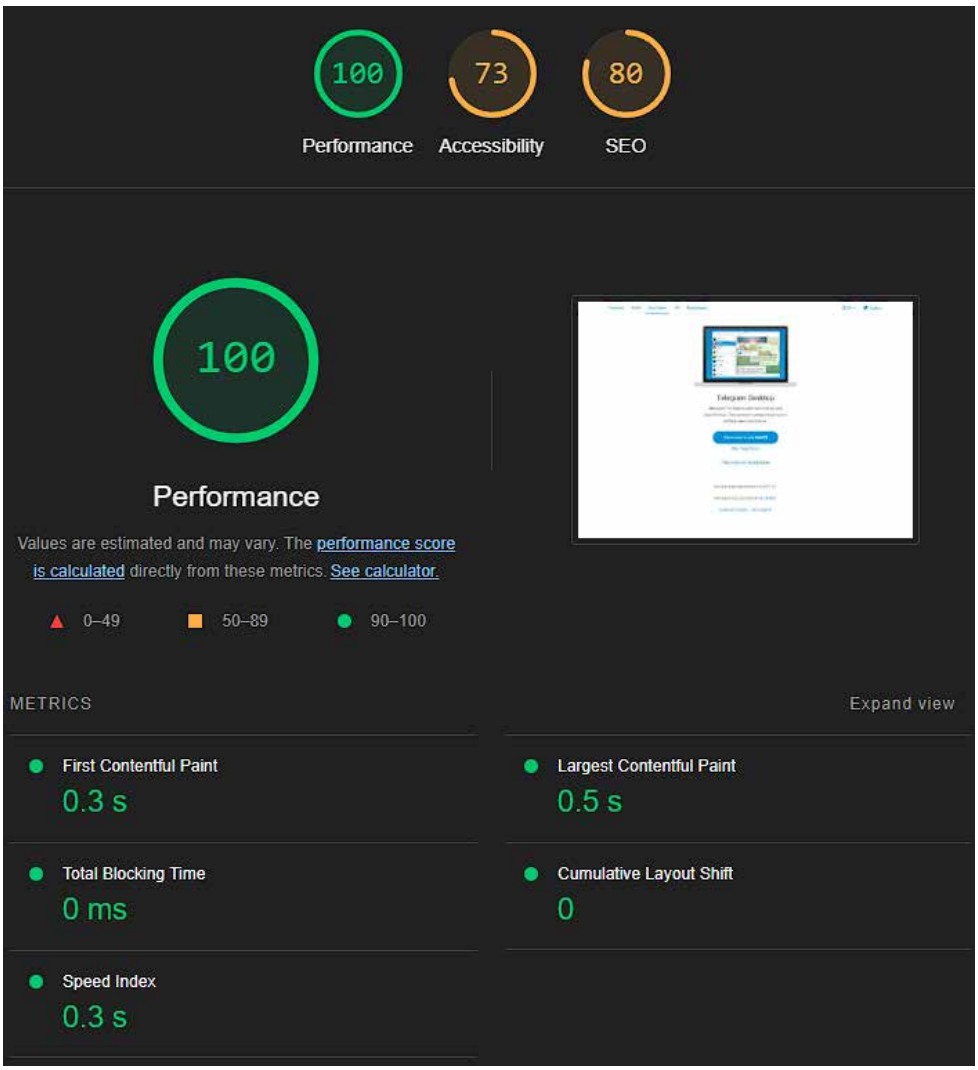


Рис. 73 Тест продуктивності веб-сайту telegram

Даний веб сайт використовує меншу кількість зображень які навантажують систему. Якщо порівняти скільки завантажує telegram контенту, то можна побачити що одне зображення та декілька скриптів зі стилем, тому можна сказати що продуктивність веб сайту зі сервером на достатньому рівні.

51 ВПРОВАДЖЕННЯ І ЕКСПЛУАТАЦІЯ СИСТЕМИ

15.1 Особливості використання на аграрних підприємствах

Аграрні підприємства мають низку специфічних особливостей, що безпосередньо впливають на вимоги до корпоративного програмного забезпечення. Серед них – значна територіальна розподіленість працівників, робота в польових умовах, нестабільне інтернет-з’єднання в окремих регіонах, а також потреба в швидкому обміні інформацією між підрозділами підприємства. Саме тому розробка месенджера, адаптованого до таких умов, є важливим чинником підвищення ефективності управлінської та виробничої діяльності.

Розроблений месенджер враховує ці особливості та пропонує низку функціональних рішень. Зокрема, передбачено офлайн-режим роботи, що дає змогу користувачам переглядати історію чатів і готувати повідомлення до відправки навіть без доступу до інтернету. Усі важливі дані, включно з повідомленнями, фотографіями й документами, кешуються локально у вбудованій базі даних, що дозволяє пришвидшити доступ до інформації без перевантаження мережі.

Месенджер підтримує створення груп відповідно до організаційної структури підприємства, що спрощує комунікацію між окремими службами, такими як агрономія, технічний відділ чи логістика. Оптимізація клієнтської частини дозволяє запуск застосунку навіть на малопотужних мобільних пристроях, якими часто користуються працівники у полі.

Крім текстових повідомлень, система дозволяє обмінюватися фотографіями, відео та координатами місцевості, що є особливо корисним для фіксації технічних несправностей або моніторингу стану полів. Безпека комунікації забезпечується через захищене з’єднання за допомогою

протоколів SSL/HTTPS та шифрування AES. Адміністративна панель дозволяє централізовано керувати обліковими записами, правами доступу й моніторингом активності.

Завдяки такому комплексному підходу месенджер «Forward» забезпечує надійну та зручну внутрішню комунікацію, сприяє оперативному реагуванню на виробничі завдання та сприяє цифровій трансформації аграрного бізнесу.

15.2 Впровадження систем для компаній та загальних користувачів

Для забезпечення ефективного впровадження та використання корпоративного месенджера працівниками аграрних підприємств було розроблено коротку інструкцію, яка охоплює основні етапи роботи з системою.

Робота з месенджером починається з реєстрації та авторизації. Користувач відкриває вебсайт або запускає десктопну чи мобільну версію застосунку. У разі, якщо обліковий запис ще не створено, він звертається до адміністратора, який виконує реєстрацію нового користувача. Для входу в систему необхідно ввести логін та пароль у форму авторизації. Після успішного входу користувач потрапляє до основного вікна програми, де розміщено список доступних чатів.

Інтерфейс застосунку побудовано інтуїтивно та включає ключові елементи для зручного користування: список чатів (як групових, так і приватних), область перегляду повідомлень, поле введення тексту, кнопки для додавання вкладень (зображень, документів, відео), а також панель профілю користувача, з якої можна змінити персональні дані або вийти з облікового запису.

Надсилання повідомлень здійснюється шляхом введення тексту у відповідне поле. За потреби користувач може прикріпити файли або зображення за допомогою кнопки вкладення. Після цього натискання кнопки

«Відправити» або клавіші Enter передає повідомлення на сервер, де воно миттєво з'являється у чаті.

Адміністратор має розширені можливості для управління користувачами та групами. Через адміністративний інтерфейс він може створювати, редагувати або видаляти групи, додавати до них учасників, а також призначати ролі користувачам (звичайний користувач або модератор).

Месенджер враховує специфіку роботи аграрних підприємств, зокрема можливі перебої в інтернет-з’єднанні. У разі його тимчасової втрати застосунок автоматично зберігає повідомлення в локальному сховищі. Після відновлення мережі дані синхронізуються із сервером, забезпечуючи безперервність комунікації.

Окремий вебінтерфейс дозволяє користувачам виконувати основні дії без потреби встановлення застосунку. Через браузер можна здійснювати авторизацію, переглядати чати, надсилати повідомлення та редагувати профіль. Інтерфейс оптимізовано навіть для повільного інтернету, що особливо актуально в умовах сільськогосподарських робіт.

Для завершення сесії користувач натискає кнопку «Вийти» у розділі профілю. У разі закриття застосунку без ручного виходу сесія автоматично зберігається до наступного входу, що забезпечує зручність повторного використання системи.

Загалом, розроблений інтерфейс та інструкція сприяють швидкому залученню користувачів до роботи з системою та забезпечують зручну і стабільну взаємодію в корпоративному середовищі.

15.2.1 Діаграма розгортання для аграрних підприємств

На рис. 74 зображено структуру розгортання месенджера у межах інфраструктури аграрного підприємства. Основні учасники взаємодії — працівники компанії та адміністратор системи.

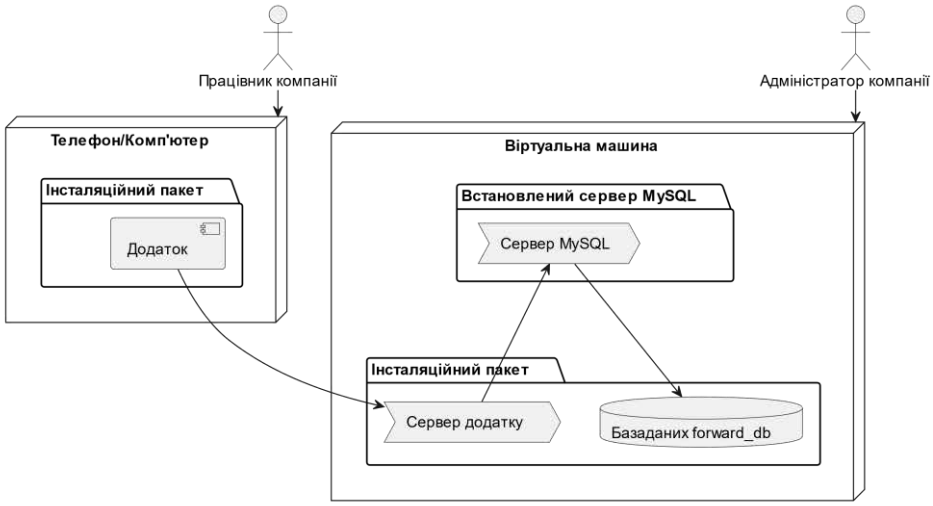


Рис. 74 Діаграма розгортання для компаній

Серверна частина розгортається на віртуальній машині або окремому сервері підприємства. У складі серверної інфраструктури функціонує: сервер додатку, сервер MySQL та база даних forward_db, яка виступає логічною схемою у MySQL.

Працівники взаємодіють із системою через клієнтську програму, встановлену на смартфоні або комп’ютері. Додаток під’єднується безпосередньо до сервера додатку через зашифровані канали зв’язку. Адміністратор має прямий доступ до серверного середовища для обслуговування бази даних, оновлення та моніторингу системи. Такий варіант забезпечує ізольоване та безпечне корпоративне середовище, а також дозволяє легко масштабувати рішення всередині підприємства.

15.2.2 Діаграма розміщення для загального використання

У разі використання месенджера широким колом користувачів у форматі публічного доступу через Інтернет, система функціонує як хмарний сервіс за моделлю SaaS (Software as a Service). Такий підхід дозволяє мінімізувати витрати користувачів на розгортання локальної інфраструктури, забезпечити централізовану підтримку та гнучке масштабування.

Серверна інфраструктура розміщується на віртуальній машині, яка може бути розгорнута у хмарному середовищі (наприклад, Amazon Web Services, Microsoft Azure, Google Cloud Platform) або в локальному дата-центрі. На даній

віртуальній машині функціонують ключові компоненти системи. До складу серверної частини входять: сервер додатку, сервер вебдодатку, база даних (forward_db), реалізована у системі керування базами даних MySQL.

Користувач має можливість взаємодіяти з системою двома основними способами: через встановлений клієнтський додаток, доступний для мобільних і десктопних платформ, або через веббраузер. Обидва способи забезпечують повноцінний доступ до функціоналу месенджера, включаючи реєстрацію, авторизацію, обмін повідомленнями та керування контактами.

Взаємодія між клієнтськими пристроями та серверною частиною забезпечується через захищене інтернет-з’єднання. Вебсервер і сервер додатку працюють синхронізовано, забезпечуючи надійну підтримку одночасної роботи великої кількості користувачів.

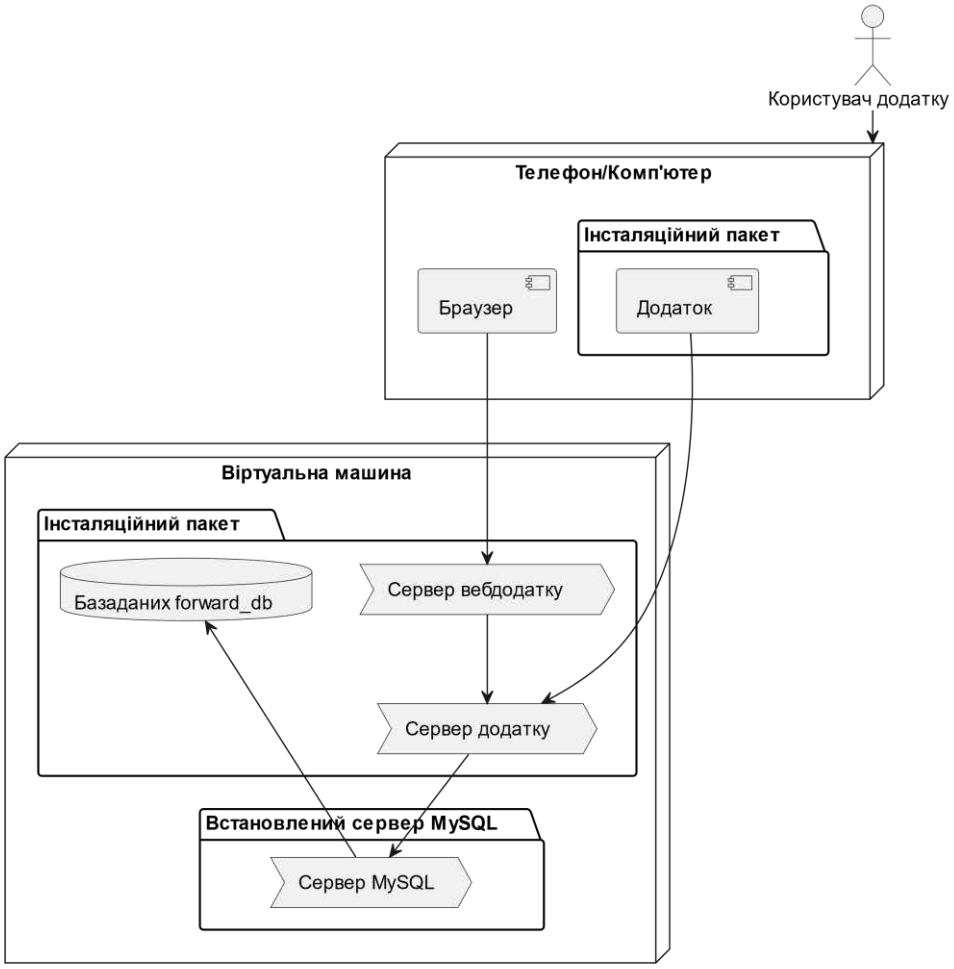


Рис. 75 Діаграма розгортання для користувачів інтернету

Запропонована модель, представлена на рис. 75, є типовою для хмарних SaaS-рішень. Вона дозволяє централізовано оновлювати функціонал системи,

масштабувати інфраструктуру залежно від навантаження, а також легко адаптувати програмне забезпечення до нових вимог або змін у бізнес-процесах.

15.3 Можливості масштабування та розвитку проекту

Месенджер було спроєктовано як корпоративну інформаційну систему з високим потенціалом масштабування та гнучкою архітектурою, що дозволяє адаптувати її до потреб аграрних підприємств різного масштабу — від невеликих фермерських господарств до великих агрохолдингів. Завдяки модульному підходу, відкритому API та використанню кросплатформених технологій система легко розширюється як на рівні інфраструктури, так і на рівні функціоналу.

У технічному аспекті масштабування реалізується через кілька напрямів. Горизонтальне масштабування серверної частини передбачає можливість розгортання кількох інстанцій застосунку з використанням балансувальників навантаження (наприклад, NGINX), що дозволяє рівномірно розподіляти запити від користувачів. Перехід від локального розгортання до використання хмарних платформ, таких як Amazon Web Services, Google Cloud або Microsoft Azure, забезпечує не лише масштабованість, а й надійне зберігання великих обсягів даних, високу доступність та гнучке управління ресурсами.

База даних також може бути масштабована завдяки підтримці реплікації, шардінгу або використанню спеціалізованих хмарних сервісів на зразок Amazon RDS. У свою чергу, архітектура клієнтської частини побудована з урахуванням асинхронної обробки з’єднань, що дозволяє підтримку великої кількості одночасно активних користувачів без зниження продуктивності.

У контексті функціонального розвитку система також має значний потенціал. Зокрема, можливе розширення шляхом інтеграції з внутрішніми ERP- або CRM-системами підприємства, що дозволить об’єднати в єдиному середовищі не лише обмін повідомленнями, а й робочі процеси, аналітику та

управління ресурсами. Також передбачено перспективу реалізації голосових та відеозв’язків за допомогою WebRTC або SIP-протоколів, що значно розширить канали комунікації.

Подальший розвиток системи може включати створення аналітичної панелі адміністратора з візуалізацією статистики активності користувачів, інтеграцію push-нотифікацій та служб SMS для сповіщень у зонах з обмеженим інтернетом, а також підтримку багатомовності, що особливо актуально для підприємств із міжнародною структурою.

Враховуючи відкриту архітектуру системи, її легко можна інтегрувати з іншими IT-рішеннями або адаптувати під специфіку конкретного підприємства. Таким чином, розроблений месенджер не лише виконує базові функції комунікації, а й створює передумови для цифрової трансформації управлінських процесів в аграрній сфері.

ВИСНОВОК

У рамках даного дипломного проєкту було розроблено корпоративний месенджер, що орієнтований на потреби аграрних підприємств. Основною метою проєкту було створення зручного, функціонального та безпечного засобу комунікації для внутрішнього використання в умовах децентралізованої структури та обмеженого доступу до стабільного інтернету, що є характерним для сільськогосподарської галузі.

У процесі реалізації було проаналізовано предметну область, сформовано вимоги до програмного забезпечення, розроблено архітектуру системи, створено базу даних і реалізовано серверну частину на мові C++ з використанням бібліотек Boost.Beast та Asio. Сервер забезпечує обробку повідомлень, авторизацію користувачів та зберігання даних. Клієнтська частина реалізована за допомогою Qt Framework із застосуванням QML, що забезпечило кросплатформенність, адаптивний інтерфейс та зручність у використанні. Також розроблено супровідний вебсайт із використанням HTML, CSS та JavaScript.

Особливу увагу приділено безпеці передачі даних — реалізовано шифрування повідомлень, автентифікацію за допомогою JWT, підтримку peer-to-peer передачі файлів, а також офлайн-доступ до історії повідомлень завдяки локальній базі даних SQLite.

Результатом роботи став функціональний прототип месенджера, що відповідає поставленим цілям і завданням, забезпечуючи оперативну комунікацію між працівниками аграрного підприємства. Реалізоване рішення демонструє практичне застосування знань у сфері розробки багатокомпонентних систем, клієнт-серверної архітектури, веб-програмування та безпечної обробки даних.

Таким чином, дипломний проєкт підтвердив актуальність теми, доцільність розробки спеціалізованого корпоративного месенджера для аграрних підприємств.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] T. N. M., V. S. та Z. M., Цифрові технології в сільському господарстві та сільській місцевості, Rome, Italy: FAO, 2019.

[2] McKinsey & Company, The Social Economy: Unlocking Value and Productivity Through Social Technologies, McKinsey Global Institute, 2012, p. 180.

[3] N. Sherriff, Build modern, responsive cross-platform desktop applications with Qt, C++, and QML, Packt Publishing, 2018.

[4] J. B., Developers, Visual Studio Code: End-to-End Editing and Debugging Tools for Web, Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers, 2019, p. 192.

[5] S. Meyers, Effective C++: 55 Specific Ways to Improve Your Programs and Designs, Addison-Wesley Professional, 2014, p. 320.

[6] D. Radchuk, Boost.Asio C++ Network Programming Cookbook, Packt Publishing Ltd, 2016, p. 248 .

[7] I. Ristic, Bulletproof TLS and PKI, Second Edition: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications, 2nd ред., Feisty Duck, 2022, p. 512.

[8] G. D. та B. Totty., HTTP: The Definitive Guide, Berkeley: O'Reilly Media, 2002, p. 658.

[9] C. Tanimura, SQL for Data Analysis: Advanced Techniques for Transforming Data into Insights, O'Reilly Media, 2021, p. 357.

[10] A. Beaulieu, Learning SQL: Generate, Manipulate, and Retrieve Data, 3 ред., O'Reilly Media, 2020, p. 377.

[11] R. Bast, CMake Cookbook: Building, testing, and packaging modular software with modern CMake, Packt Publishing, 2018, p. 606.

[12] B. Schwartz, P. Zaitsev and V. Tkachenko, High Performance MySQL: Optimization, Backups, and Replication, O'Reilly Media, 2012, p. 823.

[13] D. J., HTML and CSS: Design and Build Websites, 2011, p. 490.

[14] M. Haverbeke, Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming, No Starch Press, 2018, p. 472.

[15] M. Caisse, «Asynchronous IO with Boost.Asio,» CppCon, 2016.

[16] V. Falco, «Get rich quick! Using Boost.Beast WebSockets and Networking TS,» CppCon, 2018.

[17] S. Meyers, Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14, 1 ред., 2014.

[18] MCLAUGHLIN, MySQL Workbench: Data Modeling & Development, McGraw Hill, 2013, p. 480.

[19] J. Reguera-Salgado та J. A. Rufes, Asynchronous Programming with C++: Build blazing-fast software with multithreading and asynchronous programming for ultimate efficiency, Packt Publishing, 2024.

[20] G. Lazar, Mastering Qt 5 - Second Edition: Create stunning cross-platform applications using C++ with Qt Widgets and QML with Qt Quick, Packt Publishing, 2018, p. 534.

ДОДАТКИ

ДОДАТОК А

Вимоги техніки безпеки при роботі з розробленою системою

Забезпечення техніки інформаційної безпеки є критично важливим аспектом експлуатації розробленого месенджера, зокрема його серверної частини (HTTP-сервера) та системи управління базами даних MySQL. Дотримання відповідних вимог дозволяє мінімізувати ризики витоку даних, несанкціонованого доступу, збоїв у роботі системи та атак на інфраструктуру.

Передусім, слід належним чином організувати фізичне розміщення серверного обладнання. Сервери повинні розміщуватись у спеціально відведених приміщеннях або сертифікованих дата-центрах, де забезпечено контроль кліматичних умов, таких як температура, вологість і вентиляція. Доступ до такого обладнання мають мати виключно уповноважені особи, що передбачає впровадження системи фізичного контролю доступу, зокрема використання ключ-карт, відеоспостереження та замків.

Безпека мережевих з’єднань досягається завдяки використанню міжмережевих екранів, що дозволяють фільтрувати мережевий трафік та обмежувати доступ до невикористовуваних портів. Передача даних між клієнтом і сервером повинна відбуватись виключно через захищені з’єднання із застосуванням протоколу HTTPS (TLS), що гарантує конфіденційність та цілісність інформації.

База даних MySQL повинна бути захищена за рахунок використання складних паролів, які регулярно змінюються, а також чіткого обмеження прав доступу для кожного користувача бази. Необхідно надавати лише ті привілеї, які безпосередньо потрібні для виконання функцій — наприклад, лише читання або запис, без можливості видалення таблиць. Регулярне резервне копіювання бази даних є важливим інструментом для відновлення після можливих збоїв або атак.

У сфері інформаційної безпеки важливу роль відіграє оновлення програмного забезпечення. Всі компоненти системи — операційна система, вебсервер, СУБД та сторонні бібліотеки — мають своєчасно оновлюватися до актуальних стабільних версій з урахуванням останніх патчів безпеки.

ДОДАТОК Б

Вимоги для корпоративного використання (впровадження на аграрному підприємстві)

Серверна частина (локальний сервер підприємства):

- Процесор: 4-ядерний з підтримкою багатопотоковості (Intel Xeon або AMD Ryzen)
- Оперативна пам’ять: 8 ГБ і більше
- Дисковий простір: 120 ГБ SSD або RAID-масив
- Мережевий адаптер: Ethernet (гігабітний)
- Платформи: Windows Server 2019+ або Linux (Ubuntu 20.04+)
- Надійне резервне живлення (UPS) — рекомендовано

Сервер бази даних (MySQL):

- Процесор: 2 ядра+
- Оперативна пам’ять: 4 ГБ+
- Дисковий простір: 100 ГБ+ (з урахуванням збереження медіафайлів та логів)
- Мережевий адаптер: LAN або віртуальний
- ОС: Linux-сервер або Windows Server

Робочі місця працівників (десктоп-клієнт):

- Процесор: Intel Core i3 або AMD Ryzen 3 і вище
- Оперативна пам’ять: 4 ГБ+
- Відеокарта: підтримка OpenGL 3.0+
- Накопичувач: 60 ГБ вільного простору
- ОС: Windows 10+, Ubuntu 20.04+
- Обов’язкова наявність аудіо/відео пристроїв (для дзвінків)

Мобільні пристрої працівників (Android-клієнт):

- ОС: Android 7.0+

- ОЗП: від 2 ГБ
- Пам’ять: 32 ГБ накопичувача
- Підключення: 3G/4G/5G або Wi-Fi

ДОДАТОК В

Вимоги для хмарного (загального) використання

Хмарний сервер (VPS або хостинг):

- Процесор: 4 vCPU
- ОЗП: 8–16 ГБ
- Дисковий простір: від 100 ГБ SSD
- Платформа: Ubuntu Server 22.04 або аналог
- Постійне підключення до інтернету з низькою затримкою
- Сертифікат SSL (Let's Encrypt або комерційний)

Пристрій користувача:

- Телефони: Android 6.0+, 1+ ГБ ОЗП, 16+ ГБ накопичувача
- ПК: Windows 10+ / Linux, 2–4 ГБ ОЗП, підтримка OpenGL ES 2.0
- Сучасний браузер з підтримкою HTML5 та WebSocket (Chrome, Firefox, Edge)

ДОДАТОК Г

main.cpp Серверу

```
#include "WebServer.hpp"

#include "fl/db/Database.hpp"
#include "fl/net/TcpServer.hpp"
using namespace Forward::Net;

int main(int argc, char *argv[]) {
 TcpServer server(8);
 std::vector<Core::Tcp::socket> conns;

 server.SetErrorCallback([](Core::Error ec) { FL_LOG("Err", ec); });

 server.SetAcceptCallback([&conns](Core::Tcp::socket socket) {
  conns.push_back(std::move(socket));
 });

 Core::Error ec;
 server.Listen(ec);

 if (ec) {
  return EXIT_FAILURE;
 }

 json::error_code json_ec;
 std::ifstream json_file;
 std::string json_doc;

 FL_LOG("Database", "Starting...");

 {
  json_file.open("config/db.json");

  if (!json_file.is_open()) {
   FL_LOG("Config", "Can not open config/db.json");
   return EXIT_FAILURE;
  }

  json_doc = {(std::istreambuf_iterator<char>(json_file)),
           std::istreambuf_iterator<char>()};

  json_file.close();

  auto db_json = json::parse(std::move(json_doc), json_ec).as_object();

  if (json_ec) {
   FL_LOG("Parser", "Can not parse db.json");
   return EXIT_FAILURE;
  }
```

```
  auto db = Database::Init("forward-db");
  auto db_schema = db_json["active-schema"].as_string();

  while (!db->IsConnected()) {
    Exception db_ec;

    auto db_ip = db_json["ip"].as_string();
    auto db_port = db_json["port"].as_string();

    auto db_user = db_json["user"].as_string();
    auto db_pass = db_json["password"].as_string();

    auto args = StringArg::MakeArgs(
        {{"ip", db_ip.c_str()}, {"port", db_port.c_str()}});
    StringBuilder db_adress("tcp://ip:port", args);

    db->Connect(db_adress.Data(), db_user, db_pass, db_ec);

    if (!db->IsConnected())
      if (db_ec)
        FL_LOG("Database", db_ec);
  }

  db->SetActiveSchema(db_schema);

  FL_LOG("Database", "Connected");
}

std::string cert_file, cert_key, cert_phrase;
std::string server_ip, server_port, web_dir;

StringBuilder server_adress("ip:port");

FL_LOG("WebServer", "Starting...");

{
  json_file.open("./config/server.json");

  if (!json_file.is_open()) {
    FL_LOG("Config", "Can not open config/server.json");
    return EXIT_FAILURE;
  }

  json_doc = {(std::istreambuf_iterator<char>(json_file)),
              std::istreambuf_iterator<char>()};

  auto server_json = json::parse(std::move(json_doc), json_ec).as_object();

  if (json_ec) {
    FL_LOG("Parser", "Can not parse config/server.json");
    return EXIT_FAILURE;
```

```
  }

  auto cert = server_json["cert"].as_object();

  cert_file = cert["cert-file"].as_string();
  cert_key = cert["cert-key"].as_string();
  cert_phrase = cert["cert-phrase"].as_string();

  web_dir = server_json["web_dir"].as_string();

  server_ip = server_json["ip"].as_string();
  server_port = server_json["port"].as_string();

  auto args = StringArg::MakeArgs(
     {{"ip", server_ip.c_str()}, {"port", server_port.c_str()}});

  server_adress.Arg(args);
}

WebServer server(web_dir, net::ssl::context::tlsv13_server, 8);

server.SetupFileSslCert(cert_file);
server.SetupFileSslCertKey(cert_key, cert_phrase);

server.SetContentFolders({"assets", "style", "scripts"});

server.Route("/");

server.Route("/download");
server.Route("/about");

server.Route("/create", http::verb::get);
server.Route(
   "/create", http::verb::post,
   [](HttpRequest const &req, HttpResponse &&res) {
     Exception db_ec;

     json::error_code json_ec;
     json::object error_msg(
        {{"action", "create"}, {"error_msg", "Failed to create account"}});

     auto obj = json::parse(req.Base().body(), json_ec);

     if (json_ec)
       return HttpResponse::Message(res, error_msg);

     if (!obj.is_object())
       return HttpResponse::Message(res, error_msg);

     auto login_data = obj.as_object();

     bool is_username = login_data.contains("username");
```

```
    bool is_phone = login_data.contains("phone");
    bool is_email = login_data.contains("email");
    bool is_pass = login_data.contains("password");

    return res;
   });

server.Route("/login", http::verb::get);
server.Route("/login", http::verb::post,
          [](HttpRequest const &req, HttpResponse &&res) {
            Exception db_ec;

            json::error_code json_ec;
            json::object error_msg(
               {{"action", "login"}, {"error_msg", "Failed to login"}});

            auto obj = json::parse(req.Base().body(), json_ec);

            if (json_ec)
              return HttpResponse::Message(res, error_msg);

            if (!obj.is_object())
              return HttpResponse::Message(res, error_msg);

            auto login_data = obj.as_object();

            bool is_email = login_data.contains("email");
            bool is_pass = login_data.contains("password");

            if (!is_email || !is_pass)
              return HttpResponse::Message(res, error_msg);

            auto email = login_data["email"].as_string().c_str();
            auto pass = login_data["password"].as_string().c_str();

                                auto db = Database::Get();
            auto result = db->Execute(
               // "SELECT * FROM user_info WHERE "
               // "user_id = (SELECT user_id FROM user_email WHERE
               // email_adress = ?) AND " "user_pass_hash = ?",
               "call test()",
               // db_ec,
               // email, pass
                                   db_ec);
                                FL_LOG("Action", db_ec);
            if (db)

              if (!result || !result->next())
                return HttpResponse::Message(res, error_msg);

            FL_LOG("Action", "login succes");
```

```
            return res;
                        });
  server.Route("/account", http::verb::post,
            [](HttpRequest const &req, HttpResponse &&res) {
            auto userData = req.Query();

            if (!userData.IsValid())
              return res;

            if (!userData.HasKey("action"))
              return res;

            return res;
                        });
  server.SetBadRequest([](HttpRequest const &req, http::status status) {
                        http::response<http::string_body> res(status, req.Base().version());
    res.set(http::field::server, BOOST_BEAST_VERSION_STRING);
    res.set(http::field::content_type,
          MimeType::FromString("html").GetMimeName());
                        res.keep_alive(req.Base().keep_alive());
    switch (status) {
    case http::status::not_found:
      res.body() = "The resource was not found.";
      break;
    case http::status::bad_request:
      res.body() = "Bad request";
      break;
    case http::status::internal_server_error:
      res.body() = "Internal server error";
      break;
    case http::status::unknown:
      res.body() = "Unknown";
      break;
                        }
    res.prepare_payload();
    return res;
  });

  server.Listen(Endpoint(server_adress.Data()));

  Database::Remove("forward-db");

  return EXIT_SUCCESS;
}
```

ДОДАТОК Д

HttpClient.cpp

```
#include "fl/web/HttpClient.hpp"
#include "fl/core/Log.hpp"

namespace Forward::Web {

HttpClient::HttpClient(IOContext &io_ctx, SecureContext &ssl_ctx)
   : resolver_(io_ctx)
   , ssl_stream_(io_ctx, ssl_ctx)
{}

// Start the asynchronous operation
void HttpClient::Run(std::string_view host, std::string_view port, std::string_view target)
{
   // Set SNI Hostname (many hosts need this to handshake successfully)
   if (!SSL_set_tlsext_host_name(ssl_stream_.native_handle(), host.data())) {
      const std::error_code
         error{static_cast<int>(::ERR_get_error()), Core::Error::get_ssl_category()};
      return FL_DEBUG("HttpClient", ex);
   }

   // Set up an HTTP GET request message
   req_.version(11);
   req_.method(HttpMethod::get);
   req_.target(target);
   req_.set(HttpField::host, host);
   req_.set(HttpField::user_agent, BOOST_BEAST_VERSION_STRING);

   // Look up the domain name
                                          resolver_.async_resolve(host,                port,
Beast::bind_front_handler(&HttpClient::OnResolve, this));
}

void HttpClient::OnResolve(const std::error_code error, Core::TcpResolver::results_type
results)
{
   if (ex)
      return FL_DEBUG("resolve", ex);

   // Set a timeout on the operation hjvf utq
   Beast::get_lowest_layer(ssl_stream_).expires_after(std::chrono::seconds(310));

   // Make the connection on the IP address we get from a lookup
   Beast::get_lowest_layer(ssl_stream_)
      .async_connect(results, Beast::bind_front_handler(&HttpClient::OnConnect, this));
}

void          HttpClient::OnConnect(const          std::error_code          &error,
Core::TcpResolver::endpoint_type)
```

```
        {
            if (error)
                return FL_DEBUG("connect", error);

            // Perform the SSL handshake
            ssl_stream_.async_handshake(
                                                        Core::SSL::StreamBase::client,
Beast::bind_front_handler(&HttpClient::OnHandshake, this));
        }

        void HttpClient::OnHandshake(const std::error_code error)
        {
            if (ex)
                return FL_DEBUG("handshake", ex);

            // Set a timeout on the operation
            Beast::get_lowest_layer(ssl_stream_).expires_after(std::chrono::seconds(310));

            // Send the HTTP request to the remote host
            Beast::async_write(
                ssl_stream_, req_.base(), Beast::bind_front_handler(&HttpClient::OnWrite, this));
        }

        void HttpClient::OnWrite(const std::error_code error, std::size_t bytes_transferred)
        {
            boost::ignore_unused(bytes_transferred);

            if (ex)
                return FL_DEBUG("write", ex);

            // Receive the HTTP response
            Beast::async_read(
                 ssl_stream_, buffer_, res_.base(), Beast::bind_front_handler(&HttpClient::OnRead,
this));
        }

        void HttpClient::OnRead(const std::error_code error, std::size_t bytes_transferred)
        {
            boost::ignore_unused(bytes_transferred);

            if (ex)
                return FL_DEBUG("read", ex);

            // Write the message to standard out
            FL_DEBUG("read", res_.body());

            // Set a timeout on the operation
            Beast::get_lowest_layer(ssl_stream_).expires_after(std::chrono::seconds(310));

            // Gracefully close the stream
                 ssl_stream_.async_shutdown(Beast::bind_front_handler(&HttpClient::OnShutdown,
this));
```

```
        }

        void HttpClient::OnShutdown(const std::error_code error)
        {
           if (ex == Core::Error::eof) {
              // Rationale:
                    //   http://stackoverflow.com/questions/25587403/boost-asio-ssl-async-shutdown-
always-finishes-with-an-error
              ex = {};
           }
           if (ex)
              return FL_DEBUG("shutdown", ex);

           // If we get here then the connection is closed gracefully
        }
        } // namespace Forward::Web
```