Хмельницький національний університет
Факультет програмування
та комп'ютерних і телекомунікаційних систем
Кафедра інженерії програмного забезпечення

ДИПЛОМНИЙ ПРОЕКТ

Програмна система обміну повідомленнями
з використанням стійких алгоритмів шифрування
Назва теми

Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-прфоесійна програма «Інженерія програмного забезпечення»

Шифр ДПІПЗ.170109.17.09 ПЗ

Виконав студент IV курсу група ІПЗ-17-1 _____ Є. В. Майор
Підпис Ініціали, прізвище

Керівник канд. техн. наук, доцент _____ І. В. Гурман
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормконтролер канд. техн. наук, доцент _____ Г. І. Радельчук
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

7 квітня 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Програмування та комп'ютерних і телекомунікаційних систем
Кафедра Інженерії програмного забезпечення
Освітній рівень Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ:
Завідувач кафедри
Л. П. Бедратюк
05 02 2021 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Майору Євгену Віталійовичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Програма система обміну повідомленнями з використанням стійких алгоритмів шифрування

Керівник проекту (роботи) Гурман Іван Васильович
кандидат технічних наук, доцент
Затверджено наказом ректора університету від 05.02 2021 р. № 11
2. Строк подання студентом проекту (роботи) на кафедру: 01.06.2021 р.
3. Вихідні дані до проекту (роботи) Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація системи, тестування програмної системи

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди, 18 шт.)

6. Консультанти розділів дипломного проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Радельчук Г. І., доцент кафедри ІПЗ		
Антиплагіат	Гурман І. В., доцент кафедри ІПЗ		

7. Дата видачі завдання « 05 » лютого 2021р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1 Ознайомлення з тематикою дипломного проектування (ДП), визначення та узгодження індивідуальних там ДП	01.12 – 30.12.2020	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2021	
3 Проектування програмного забезпечення	01.02 – 28.02.2021	
4 Програмна реалізація	01.03 – 10.04.2021	
5 Тестування програмного забезпечення	11.04 – 30.04.2021	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	01.05 – 25.05.2021	
7 Попередній захист ДП	Травень 2021 (згідно графіка)	
8 Перевірка ДП на плагіат, нормконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки)	26.05 – 30.05.2021	
9 Підготовка до захисту та захист ДП	з 01.06.2021	

Студент _____ Є.В. Майор
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ І.В. Гурман
Підпис Ініціали, прізвищ

АНОТАЦІЯ

Тема дипломного проекту: «Програмна система обміну повідомленями з використанням стійких алгоритмів шифрування».

Автор проекту: Майор Євген Віталійович.

Керівник проекту: Гурман Іван Васильович.

Пояснювальна записка: 144 с., 10 рис., 27 табл., 7 дод., 19 джерел.

Графічна частина: 18 презентаційних слайдів.

IP, TCP/IP, PROTOCOL, NETWORK, MESSENGER, .NET, .NET CORE, C#, WPF, XAMARIN, MVVM, MYSQL.

Мета проекту – розробка програмної системи, яка дозволяє обмінюватись повідомленнями через Інтернет. Програма забезпечить швидкість з'єднання і надійний спосіб передачі повідомлень, безпеку підключення і шифрування даних. Система має сучасний інтерфейс, який інтуїтивно зрозумілий користувачам.

У дипломному проекті визначено специфіку месенджерів; виконано порівняння архітектурних підходів; проведено аналіз методів та інструментів розробки системи, мережевих протоколів; проаналізовано використання баз даних; визначено особливість фреймворків при реалізації клієнтської та серверної частини програми; проаналізовано алгоритми шифрування.

Для реалізації програмної системи використано мову програмування C#, платформу WPF та Xamarin для побудови клієнтських додатків, систему керування базою даних MySQL.

В результаті проектування здійснена програмна реалізація системи обміну повідомленнями з використаннями стійких алгоритмів шифрування, а також проведена практична апробація отриманих результатів.

09.06.2021
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	ДПІПЗ.170109.17.09 ПЗ	Пояснювальна записка	144		
2	А4		Завдання на дипломний	1		
			проект			
3	А4		Анотація	1		
4	А4		Презентаційні матеріали	18		

Змн.	Арк.	№ докум.	Підпис	Дата	ДПІПЗ.170109.17.09.ПЗ		
Виконав		Майор Є.В.		1.06	Програмна система обміну повідомленнями з використанням стійких алгоритмів шифрування	Літ.	Арк. 1 / Акрушів 1
Керівник		Гурман І.В.		2.06			
Реценз.						ХНУ, ІПЗ-17-1	
Н. Контр.		Радельчук Г.І.		3.06			
Затверд.		Бедратюк Л.П.		4.06	Відомість документів		

ЗМІСТ

					ДПІПЗ.170109.17.09.ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Виконав.		Майор Є.В.		1.06	Програмна система обміну повідомленнями з використанням стійких алгоритмів шифрування	Літ.	Арк. 4 / Акрушів 144
Керівник.		Гурман І.В.		2.06			
Реценз.							
Н. Контр.		Радельчук Г.І.		3.06	Пояснювальна записка	ХНУ, ІПЗ-17-1	
Затверд.		Бедратюк Л.П.		4.06			

ВСТУП

Сьогодні спілкування досягло найвищого технологічного рівня. Саме прогрес у технологіях, дозволив людям надсилати повідомлення за секунди. Нам доступна можливість спілкуватись в режимі реального часу на великій відстані один від одного, що раніше було неможливе. Для цього використовуються спеціалізоване програмне забезпечення (ПЗ). Такі програмні системи досить комфортні у використані, але в зв'язку із специфікою роботи таких додатків, питання конфіденційності спілкування, поступово стало на перше місце. Безпека і надійність передачі повідомлень, для користувачів, займає перше місце при виборі додатку.

Оскільки, не усе наявне програмне забезпечення виконує надійний захист даних, відповідає вимогам до таких систем або вимагає від потенційного користувача додаткових вкладів для користування цим сервісом, потреба у простому, але надійному додатку, який працюватиме на найпопулярніших платформах, досить висока. Саме тому, я вирішив розробити програмну систему, яка задовільнить ці потреби та буде доступна для користувачів.

Актуальність теми дипломного проекту полягає у створенні програмної системи, яка дозволить безпечно обмінюватись повідомленнями. Простий та інтуїтивний користувацький інтерфейс, повинен надати комфортні умови для спілкування. Програмну систему можна встановити, як на мобільний телефон, так і на робочий комп'ютер, що зможе задовільнити потреби користувачів.

Мета проекту – розробити програмну систему обміну повідомленнями з використанням стійких алгоритмів шифрування, яка буде включати в себе: серверний додаток, додаток на мобільний персональний комп'ютер.

Щоб задовільнити вимоги, потрібно вирішити наступні завдання:

– виконати аналіз предметної області, який включає: дослідження способів передачі повідомлень та аналіз існуючих рішень;

– визначити вимоги до програмного забезпечення;

– провести проектування програмної системи;

– порівняти типи архітектур, які можуть вирішити проблему, і вибрати шаблони проектування системи;

– виконати декомпозицію із детальним проектування модулів;

– визначитись із засобами реалізації системи;

– провести проектування програмних модулів;

– розробити програмну систему;

– провести тестування із кінцевим аналізом результатів.

Розроблюваний програмний продукт, може бути корисний для звичайних людей, які хочуть безпечно спілкуватися із своїми друзями. Можливе використання у IT-компаніях, як локальний месенджер.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Програмні системи обміну повідомленнями – це спеціальні клієнтські додатки, які встановлюються на особисті пристрої користувачів. Керує обміном даними сервер, який знаходиться на стаціонарній машині і з’єднується із клієнтом через мережу Інтернет.

Такі програмні системи називаються месенджери. Обов’язковою умовою для їх використання є: підключення до мережі Інтернет; наявність аналогічної системи у кінцевого отримувача повідомлення.

Обмін даними має наувазі передачу: текстових, файлових повідомлень. Ці дані повинні надійно шифруватись.

Система обміну повідомленнями використовує протокол. Протоколи можуть бути серверними та безсерверними. Серверний протокол, це коли месенджер не працює самостійно, а встановлює зв’язок із сервером, тобто виконує функцію клієнтської програми.

Раніше використовували електрону пошту для обміну повідомленями. Але поступово, обмін повідомленнями перейшов в режим реального часу або онлайн обмін. Через специфіку роботи електроної пошти, повідомлення зберігається на сервері у поштовій скринці, для того щоб отримати повідомлення, користувачеві потрібно перевірити свою електронну пошту і відкрити потрібний лист.

Месенджери замінили текстові повідомлення та стали чудовою альтернативою голосовим дзвінкам, оскільки, це позбавляє від необхідності поповнювати мобільний телефон. Щоб користуватись додатком, обов'язково потрібне підключення до Інтернету.

Отже, об’єктом розробки є месенджер, програма, яка дозволяє обмінюватися повідомленнями, в яких повідомлення надсилаються майже відразу. Користувачі спілкуються між собою, обмінюючись текстовими

повідомленнями. Залежно від функціональних можливостей, також можливий: обмін телефонними дзвінками або розмова шляхом відоезв'язку, відправка картинки або стікера, пересилання файлів різного формату.

Месенджер – це клієнтська програма, яка найчастіше встановлюється на телефоні користувача. Як уже зазначалося, щоб зареєструватися, потрібно мати номер телефону. Відразу після налаштування та реєстрації облікового запису, користувач має змогу почати спілкуватися з іншими користувачами, у яких встановлений той самий месенджер.

Однак, повністю відмовитися від мобільного зв'язку не є можливим. Щоб зареєструвати обліковий запис, у більшості месенджерів, вам потрібно буде підтвердити номер телефону.

Великою перевагою месенджерів є те, що користувачі можуть залишатися на зв'язку цілий день, завжди можна відкрити додаток і перевірити чи надійшли нові повідомлення.

За кожною програмою цього типу є мережа обміну повідомленнями, яка також є частиною концепції обміну повідомленнями. Це може бути мережа всередині компанії або глобальна мережа. Через те, що цих мереж нині досить багато, не всі месенджери сумісні. Це означає, що якщо у вашого партнера інший месенджер, ніж ваш, ви не зможете зв’язатися зі своїм партнером.

Потрібні різні мережі обміну повідомленнями, оскільки вони працюють по різному і вони немають прямого зв’язку. Програми створюються окремими компаніями або групами розробників і мають власні сервери та протоколи, функції та правила використання.

Проведено первиний аналіз предметної області, визначено суть поняття обмін повідомлення і системи якими відбувається передача повідомлень. Визначено і частково описано об’єкт розробки.

Розглянемо найпопулярніші Інтернет-месенджери, усі вони працюють із підключенні до Інтернету. Месенджер може бути встановлений не тільки на телефон, а і на комп'ютер.

1.2 Огляд існуючого програмного забезпечення

Розпочнемо із визначення критеріїв аналізу наявних програмних систем. Зараз доступна велика кількість програм обміну повідомленнями, але досліджуватись будуть лише самі популярні.

Щоб описати програмний продукт, буде коротко сказано про розробників, переваги та недоліки систем, способи захисту, доступність на різних платформах. В кінці будуть підведені підсумки дослідження.

IMessage – розроблений компанією Apple, програмне забезпечення для обміну повідомленнями. Розпочнемо із опису плюсів та мінусів технології.

До переваг можна віднести:

– додаток за замовчуванням встановлений на усі пристрої iPhone та Mac, що говорить про те, що система працює стабільно;

– простий та зрозумілий інтерфейс;

– необмежені повідомлення;

– можливість передавати файли різних форматів.

До недоліків можна віднести:

– необхідність у номері мобільного телефону і продукції Apple;

– при помилці надсилання повідомлення, воно надійде до кінцевого отримувача, як SMS.

Щодо безпеки відправки повідомлень, IMessage використовує наскрізний алгоритм шифруванням, але отримати доступ до даних все таки можливо, тому ступінь захисту система оцінюється, як середня.

Signal – безкоштовна програмне забезпечення для обміну повідомленнями і можливістю дзвінків по протоколу IP. Додаток доступний для операційних систем: Android, IOS, macOS, Linux.

До переваг можна віднести:

– відправка текстових повідомлень;

– можливість спілкуватись шляхом Інтернет-телефонії;

– простий та зрозумілий інтерфейс;

– створення секретних чатів.

До недоліків можна віднести: необхідність в обов’язковому встановленні додатку на телефон, щоб користуватись ним на комп’ютері, відкритий код.

Щодо безпеки, система Signal використовує наскрізний алгоритм шифрування OWS. Алгоритм шифрування використовується іншими розробниками схожих програмних систем. Ступінь захисту системи висока.

Skype – програмне забезпечення, яке дозволяє обмінюватись текстовими повідомленнями, виконувати голосові або відеодзвінки між додатками, а також можливість телефонувати на мобільні телефони.

До переваг можна віднести:

– необмежений обмін текстовими повідомленнями;

– голосові та відео дзвінки;

– групові чати;

– дзвінки на мобільний номер;

– передача файлів різного формату.

Додаток безкоштовний та працює на операційних системах: Windows, Android, macOS, IOS, Linux та інших.

Щодо безпеки, Skype використовує стійкий алгоритм шифрування AES-256, який використовується для голосових, телефоних і текстових повідомлень, рівень безпеки оцінюється, як високий.

Facebook Messenger – програмне забезпечення обміну повідомленнями та відео. Програмний продукт розроблений компанією Facebook і використовує відкритий протокол MQTT. Схожіть із IMessage полягає у тому, що цей програмний продукт інтегрований у іншу продукцію Facebook, а саме, у їх соціальну мережу. Додатки працюють на операційних системах: Windows, IOS, Android, Windows Phone.

До переваг можна віднести:

– відправка текстових повідомлень;

– контрольоване сповіщення про надходження нових повідомлень;

– пошук користувачів;

– обмін файлами різного формату.

До недоліків можна віднести, велике використання оперативної пам’ять, що веде за собою зменьшення часу роботи мобільних пристроїв.

Щодо безпеки, Facebook Messanger було присвоєно звання самого незахищеного месенджера, слабкий захист користувацьких даних.

Viber – програмне забезпечення для обміну повідомленнями, можливістю Інтернет-телефонії. Щоб виконати авторизацію у додатку, потрібно мати номер мобільного телефону, програма також передає вміст телефонної книги у список чатів. Додатки працюють на операційних системах: Windows, IOS, Android, Linux, macOS.

До переваг можна віднести:

– необмежена відправка повідомлень;

– можливість надсилати файли різного формату;

– редагування повідомлень;

– підтримка самих популярних операційних систем.

До недоліків можна віднести, користувач отримує повідомлення від будь-кого, що надало можливість надходження великої кількості спаму.

Щодо безпеки, Viber використовує алгоритм шифрування, власної реалізації, Double Ratchet, ступінь безпеки оцінюється, як висока.

WhatsApp – програмна система обміну повідомленнями, де можна спілкуватися головими або відео повідомленнями. Клієнт доступний на платформах: Android, IOS, Windows, MacOS та як веб-додаток.

До переваг можна віднести:

– надсилання необмеженої кількіості повідомлень;

– можливість виконувати відео або голосові дзвінки;

– перенесення контактів із вашої телефоної книги;

– передача файлів;

– зрозумілий програмний інтерфейс;

– ведення історії листування.

Основним недоліком, є необхідність мобільного телефону для створення облікового запису. Також погано працює синхронизація між різними додатками.

Щодо безпеки, WhatsApp захищений наскрізним алгоритмом шифрування Double Ratchet, який реалізований компанією Signal, що виводить безпеку спілкування на високий рівень.

Telegram – месенджер, за допомогою якого ви можете надсилати повідомлення. Клієнтські програми Telegram доступні для Windows, MacOS, iOS, Windows Phone, Android.

Користувач може надсилати повідомлення та обмінюватися фотографіями, іншими даними різного формату, голосовими та відеоповідомленнями, файлами всіх видів та аудіо і відеодзвінками.

До переваг можна віднести:

- обмін текстовими повідомленнями;
- можливість дзвінків користувачам;
- передача і зберігання файлів різного формату;
- створення групових чатів;
- простий та зрозумілий інтерфейс;
- секретні чати;
- створення спеціальних каналів;
- можливість перегляду відео у додатку;
- інтегрована технологія ботів.

До недоліків Telegram, можна віднести наявність інформації про користувача, номер мобільного телефону.

Щодо безпеки, Telegram розробив власний протокол MTProto, що використовує пару алгоритмів шифрування. Для різних операцій, використовуються різні алгоритми шифрування: RSA-2048 та DH-2048, що робить оцінку безпеки високою.

Тепер, враховуючи основні переваги та недоліки описаних програмних систем, робимо висновок, що спільними перевагами усіх систем є наступні:

– простий та зрозумілий інтерфейс;

– необмежений обмін повідомленнями;

– безпека передачі даних;

– пошук користувачів.

Проаналізувавши недоліки наявних програмних систем, можна сказати, що програмна система повинна бути доступна усім користувачам і не потребувати додаткових вкладень для користування нею, мати великий рівень захисту.

Отже, був проведений огляд наявного програмного забезпечення, яке відповідає темі дипломного проекту, описані критерії їх оцінювання та виведені спільні переваги і недоліки для подальшого визначення вимог.

1.3 Визначення вимог до ПЗ та постановка задачі

Провівши дослідження та аналіз предметної області і подібних програмних продуктів, визначаємо вимоги до програмного забезпечення. Для початку опишемо основні умови, при яких програмна система повинна працювати, а саме функціональні вимоги.

Обов’язкові умови для роботи месенджера:

– у іншого користувача встановлена така ж програма для спілкування, переписка між користувачами різних програм неможлива;

– підключення до мережі Інтернет;

– пристрій повинен відповідати мінімальним вимогам технічного стану.

Основних недоліків оглянутих систем для обміну повідомленням є:

– складний інтерфейс;

– необхідність номеру телефону для реєстрації;

– надсилання повідомлень якщо користувач не онлайн.

Отже, при розробці технічного завдання потрібно врахувати і виправити недоліки, які є у інших програмних системах. Тепер перерахуємо основні плюси розгнянутих додатків:

– необмежена відправка текстових повідомлень;
– простий і зрозумілий дизайн інтерфейсу програми;
– підтримка великої кількості платформ;
– пошук інших користувачів;
– сповіщення про нові повідомлення.

Також непоганим плюсом, буде можливіть ведення історії повідомлень і їх редагування та відновлення після входу з іншого облікового запису.

Оскільки, програмне забезпечення, що розробляється, повино мати новий протокол для передачі даних через Інтернет і нові клієнтські додатки. Потрібно розробити серверну програму, яка повина виконувати відведену їй роботу, а саме керувати комунікацією клієнтів.

Серверний додаток повинен мати зрозумілий і простий дизайн інтерфейсу, в його роботу входить:

– робота з базою даних;
– реєстрація та авторизація клієнтів;
– отримання та відправлення даних клієнтам.

Серверна программа має певні технічні обмеження, а саме: потрібно використовувати більш потужну апаратну систему, потрібем більший об’єм пам’яті, оперативної і загальної.

Програма буде обробляти велику кількість запитів до неї і відповідно багатопоточність буде стояти на першому місці. Також у вікні програми можна слідкувати за станом роботи програми і спострігати за підключення клієнтів.

Сервер після запуску, повинен працювати завжди, за винятком непердбачуваних технічних несправностей.

Діаграма варіантів використання, є першим концептуальним поданням системи під час її проектування та розробки. Ця діаграма складається з акторів, випадків використання та взаємозв’язків між ними. Поширені елементи позначення також можуть бути використані при створенні діаграми: примітки та механізми розширення.

Для представлення сценаріїв використовують акторів, які відображають об’єкт, вони взаємодіють із системою, варіантом використання. Ці варіанти використання унікальні для кожної системи і витікають із функціональних вимог до програмного продукту.

Таблиця 1.1 – Опис акторів

Актор	Короткий опис
Користувач	Виконує реєстрацію або авторизацію в системі, шукає інших користувачів у системі, відправляє їм повідомлення, отримує повідомлення користувачів.
Сервер	Реєструє або авторизує користувачів звіряючись із базою даних, обробляє користувацькі запити, отримує дані, відправляє дані користувачів.

Спочатку побудуємо контекстну діаграму, на якій відображаються основні варіанти використання системи, а потім для кожного з них побудуємо діаграми декомпозиції. Діаграма варіантів використання показана на рисунку 1.1. Опис варіантів використання приведений у таблиці 1.2.

Таблиця 1.2 – Основні варіанти використання

Варіант використання	Актори	Ціль
Головні варіанти використання	Користувач, Сервер	Провести огляд варіантів використання

Проведемо декомпозицію варіантів використання: реєстрація; авторизація; пошук користувачів; відправка повідомлення; отримання повідомлення; редагування профілю. Розлянемо декомпозицію варіанту використання реєстрації. Графічне зображення представлено на рисунку А.1, опис діаграми показаний у таблиці 1.3.

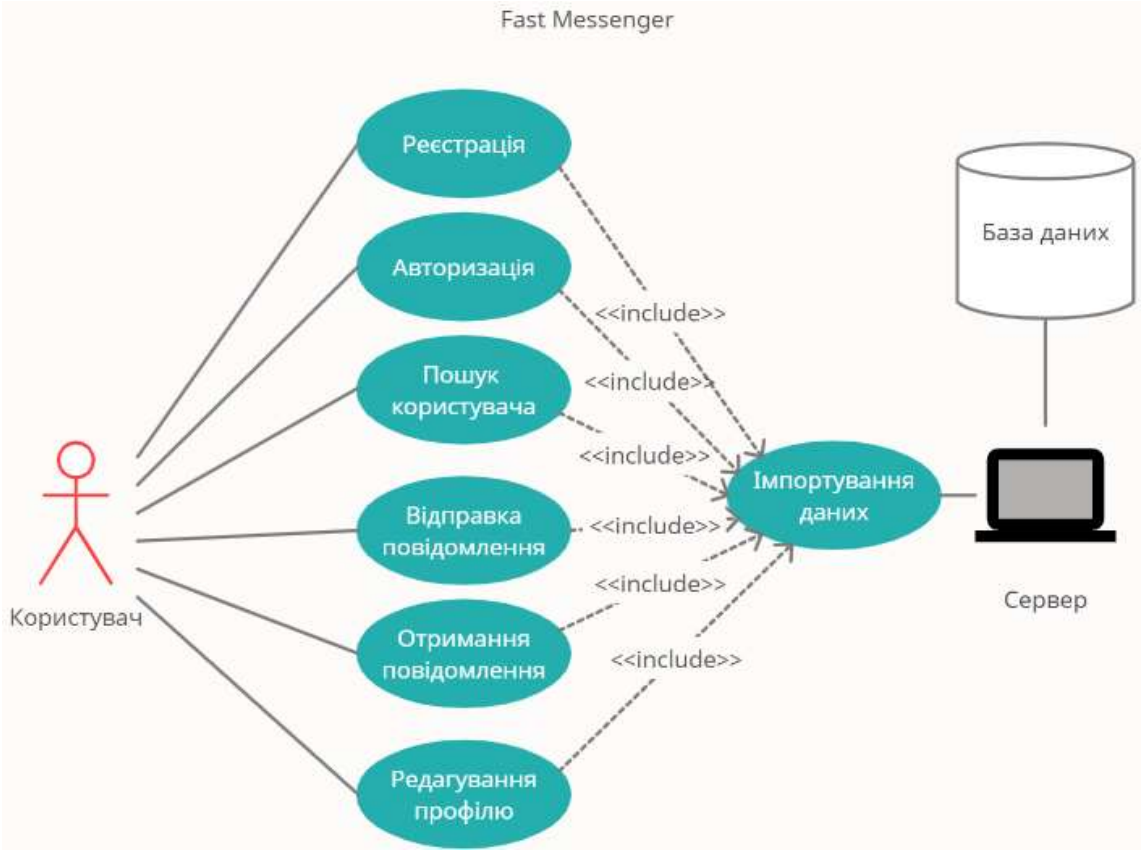


Рисунок 1.1 – Головна діаграма варіантів використання

Таблиця 1.3 – Варіант використання реєстрація

Ціль	Провести реєстрацію користувача
Короткий опис	Користувач вводить дані для реєстрації облікового запису у систему. Сервер отримує дані від користувача і проводить спробу реєстрації у базі даних і відправляє відповідь користувачу
Посилання на інши варіанти використання	Реєстрація включає наступні варіанти: відправка даних, введення даних, отримання відповіді, перевірка коректності. Обробка даних включає, отримання даних.

Діаграма варіанту використання авторизації користивача, графічно зображена на рисунку А.2, а її опис показаний у таблиці 1.4.

Діаграма варіанту використання відправки повідомлення, графічно зображена на рисунку А.3, а її опис показаний у таблиці 1.5.

Таблиця 1.4 – Варіант використання авторизація

Ціль	Провести авторизацію користувача
Короткий опис	Користувач відправляє дані для авторизації. Сервер отримує дані від користувача і перевіряє наявність такого облікового запису, після чого відправляє відповідь.
Посилання на інши варіанти використання	Авторизація включає наступні варіанти: відправка даних, введення даних, обробка даних, перевірка коректності, імпортування даних. Актор сервер виконує: імпортування даних, оброку даних, відправку даних.

Таблиця 1.5 – Варіант використання відправка повідомлення

Ціль	Відправити повідомлення певному користувачу
Короткий опис	Користувач відправляє дані на сервер із позначкою повідомлення. Сервер отримує дані від користувача і перенаправляє повідомлення до кінцевої точки або вносить користувача у список очікування до відправки.
Посилання на інши варіанти використання	Відправка повідомлення включає: написання повідомлення, перевірку повідомлення, відправку повідомлення, поновлення записів, отримання відповіді. Обробка даних влючає, отримання даних. Відправка повідомлення може розширити внесення користувача у чергу. Перевірка користувача включає: перевірку підключення, перевірку у БД.

Діаграма варіанту використання реєстрації користувача, яка задовільняє умову безпечної взаємодії клієнта із серверним додатком, графічно зображена на рисунку А.4, а її описа показаний у таблиці 1.6.

Таблиця 1.6 – Варіант використання пошук користувачів

Ціль	Провести пошук користувачів
1	2
Короткий опис	Користувач відправляє ключові слова для пошуку потрібного йому користувача і очікує результату пошуку. Сервер отримує дані від користувача, проводить пушук в БД і відправляє відповідь.

Кінець таблиці 1.6

1	2
Посилання на інши варіанти використання	Пошук включає наступні варіанти: введення даних, відправка даних, отримання відповіді, відображення результату. Обробка даних включає отримання даних. Сервер відтворює варіант використання відправка відповіді.

Діаграма варіанту використання отримання повідомлення, графічно зображена на рисунку А.5, а її описа показаний у таблиці 1.7.

Таблиця 1.7 – Варіант використання отримання повідомлення

Ціль	Отримати повідомлення від іншого користувача і відобразити його
Короткий опис	Користувач отримує повідомлення від іншого користувача і після обробки даних поновлює записи.
Посилання на інши варіанти використання	Отримання включає: обробка даних, поновлення записів. Перевірка включає: перевірка у БД, актор сервер виконує відправку повідомлення.

Діаграма варіанту використання редагування профілю, графічно зображена на рисунку А.6, а її описа показаний у таблиці 1.8.

Таблиця 1.8 – Варіант використання редагування профілю

Ціль	Редагувати профіль: ім’я, пароль
Короткий опис	Користувач проводить зміну свого облікового запису і відправлає дані серверу. Сервер отримує дані від користувача і поновлює інформацію у БД.
Посилання на інши варіанти використання	Редагування профілю включає: відправка даних для редагування, отримання відповіді, поновлення записів. Обробка даних включає отримання даних. Перевірка користувача включає порівняння із БД.

Проведено аналіз існуючого програмного та інформаційного забезпечення предметної області, в результаті якого, визначені функціональні та експлуатаційні вимоги до програмної системи, недоліки подібного програмного забезпечення, виведені основні плюси, які і стали вимогами для розробки програмної системи, визначені типи додатків. Описані діаграми варіантів використання програмної системи. І в результаті, написане технічне завдання на програмну систему, яке знаходиться у додатку Г і відповідає усім вимогам.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір типу архітектури та шаблонів проектування

Розпочнемо із аналізу та порівняння клієнт-серверної архітектури із Peer-to-peer архітектурою. В ході аналізу, визначимо основні недоліки та переваги цих архітектур і зробимо вибір необхідної.

Потім проведемо опис шаблону проектування додатків і відповідно до нього, визначимося із технологіями і платформами, які будуть використовуватись при реалізації програмної системи.

Основним елементом програмного продукту, що розробляється, є взаємодія клієнта із сервером. Завдяки цій взаємодії клієнт може зв'язуватись із сервером і обмінюватись із ним повідомленнями, сервер отримує повідомлення від клієнта і визначає, яким чином ними розпорядитися. Такий тип архітектури називається «Клієнт-сервер», зазвичай програми розміщені на різних комп'ютерах. Взаємодія проводиться через мережеві протоколи. На клієнт-серверній архітектурі побудовані сайти і Інтернет-сервіси, але також цю архітектуру використовують і десктоп додатки, вони так само передають дані через Інтернет. Важливою частиною цієї архітектури являється база даних, вона зазвичай встановлена на сервері і у ній зберігаються дані про клієнта. Можна сказати, що клієнт робить запити до бази даних через сервер.

Ще одним видом архітектури, яка може підійти для реалізації проекту, є системна архітектура «Peer-to-peer», архітектуру скорочено називають P2P. Основою цієї архітектури є мережа вузлів, які мають рівні права один між одним. Дана архітектура забезпечує баланс навантаження, ресурси клієнтів об'єднуються і взаємодіють в децентралізованій системі. В даній архітектурі клієнти взаємодіють напряму один з одни. Користувачі одночасно використовують і забезпечують існування мережі. Один вузол виконує частину рахункових ресурсів.

Унікальністю і великою перевагою P2P над клієнт-серверною архітектурою являється те, що мережа має можливість покращити свої показники у випадку приєднання великої кількості клієнтів, але такий підхід несе за собою велику кількість недоліків і дірок у системі захисту. Якщо брати клієнт-сервер, то при підключені великої кількості користувачів ефективність мережі зменьшується, що несе за собою необхідність в передчасному прорахуноку можливих системних і мережевих навантажень на сервер і необхідність у покращенні серверної частину, у випадку малої швидкості роботи мережі.

Позначимо основні переваги клієнт-серверної архітектури:

– клієнт і сервер є окремими додатками, їх розробка виконується окремо;

– захист даних, оскільки усі дані зберігаються на сервері і лише сервер визначає кому надавати до них доступ;

– при правильному прогнозувані, сервер може обслуговувати необхідну кількість клієнтів.

Позначимо основні переваги тимчасової мережі (P2P):

– низька вартість створення невеликої мережі;

– користувач може керувати розділеням ресурсів;

– легко встановлюється;

– відсутня необхідність в використанні серверів.

Розпишемо основні недоліки архітектури клієнт-сервер:

– вартість обладнання;

– необхідність у наймані спеціаліста для обслуговування і підтримки роботи сервера;

– якщо виникнуть технічні неполадки на сервері, то уся мережа не буде працювати.

Розпишемо основні недоліки архітектури P2P:

– низький рівень безпеки;

– передача ресурсів зазвичай відбувається без домовленості між вузлами;

– тимчасові вузли неможуть обслуговувати велику кількість підключень;

– користувачі повині адмініструвати власні комп’ютери;

– відсутність організації ресурсів, це ускладнює пошук даних;

– відсутнє централізоване місце зберігання даних.

Отже, якщо підсумувати аналіз переваг і недоліків архітектур, то робимо висновок, що клієнт-серверна архітектура, більш дорожча в обслуговувані і встановлені, але забезпечує стабільність, безпеку даних і надає безпеку користувачам, а також велику кількість клієнтів, які можуть підключитись до серверу. P2P архітектура простіша у встановлені і обслуговувані, але має низький рівень роботи, збереження, пошуку і захисту підключення і даних, обмежена кількість підключень.

Провівши аналіз цих архітектур, описавши принцип їхньої дії, виділивши основні переваги і недоліки, робимо висновок, що для нашого проекту більш підходить клієнт-серверна архітектура, оскільки вона забезпечує великий рівень захисту, збільшена продуктивність у клієнтських додатках, обслуговування на одному сервері великої кількості підключень.

Отже, було визначено архітектуру на якій буде побудована програмна система. Тепер перейдемо до опису принципів побудови таких систем для кращого її розуміння.

Клієнт-серверна архіткетура виділяє два основних компонети, які доповнюють один одного: сервер і клієнт. Архітектура клієнт-сервер показує, правила організації кооперації у мережі, тут сервер виступає поставником послуг, а клієнт замовником послуг.

Клієнт – локальний додаток, який виконує відправку запитів на сервер, щоб отримати дані або виконати певні системні дії.

Сервер – спеціальний комп’ютер, який призначений для рішення задач певного типу. Сервер виконує роботу сервісного обслуговування клієнтських запитів, надає користувачам доступ до системних ресурсів і зберігає дані у базі даних і дає змогу спілкуватись різним комп’ютерам.

Функції, які реалізовує клієнт:

– відправка запиту до серверу;

– отримання відповіді на запит;

– форма представлення візуального інтерфуйсу.

Функції, які реалізовує сервер:

– обробка клієнтських запитів;

– відправка відповіді клієнту;

– захист, надання доступу і зберігання даних.

Клієнт надсилає запит на сервер, запит обробляється і результат відправляється клієнту. Сервер обслуговує декілька клієнтів одночасно. Від клієнта може поступати декілька запитів одночасно, в такому випадку, запити формуються в чергу по пріоритетності.

Клієнт-серверна архітектура визначає правила віртуального спілкування комп'ютерів, а всі принципи взаємодії знаходяться всередині протоколу.

Протокол – угода логічного рівня, з його допомогою програми можуть обмінюватись даними. Угода встановлює один спосіб передачі повідомлень і обробки помилок.

Мережевий протокол – це правила, які дозволяють з'єднуватись пристроям і обмінюватись повідомленнями.

TCP/IP – об'єднання протоколів передачі інформації.

IP – протокол, який виконує коректну доставку повідомлень до певного адресу, інформація ділиться на пакети, які можуть постачатись неоднаково.

TCP – виконує з'єднання двох систем і забезпечує обмін даними.

Щоб збудувати клієнтс-серверну систему, існують такі концепції: слабкий клієнт, сильний сервер або тонкий клієнт, слабкий сервер. Перша концепція покладає на сервер увесь процес обробки даних, а користувач має обмежений доступ, коли сервер надсилає відповідь, дані не вимагають ніякої додаткової обробки, клієнт кооперує з користувачем так, що створює і надсилає запити, приймає вхідні результати і відтворює дані на інтерфейсі користувача, екрані. Друга концепція змушує клієнта брати на себе частину процесу обробки даних, сервер просто використовується, як сховище інформації, а процес обробки і використання даних проводить клієнт.

Щоб розробити клієнт-серверну архітектуру, використовують один із видів такої архітектури: двохрівнева архітектура та трирівнена система. Розглянемо дитальніше ці види архітектур.

Двухрівнена архітектура, як правило, у неї входить два вузли, клієнт, сервер і база даних. Сервер отримує вхідні запити і відправляє відповідь клієнту. Клієнт виконує функцію відображення графічного інтерфейсу. Особливість заключається в тому, що сервер отримує запит, опрацьовує його та відправляє назад, притому не використовуючи ніяких додаткових зовнішніх ресурсів. Схематичне представлення двохрівневої архітектури показане на рисунку 2.1.

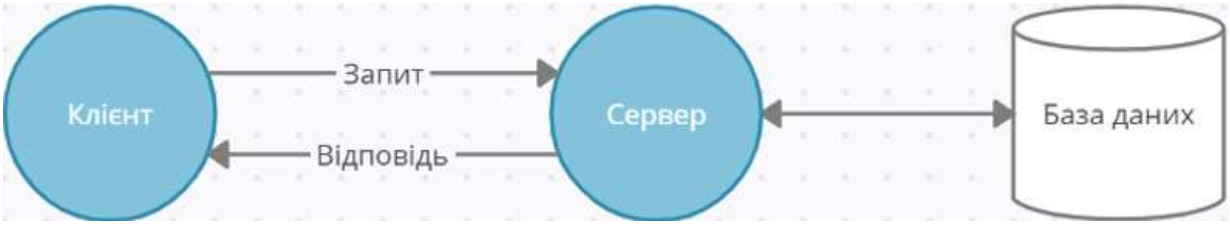


Рисунок 2.1 – Схематичне зображення двохрівневої архітектури

Трирівнева архітектура, розділена на дві або більше частини, кожна частина може виконуватися на окремому робочому комп'ютері. Частини системи взаємодіють між собою, вони обмінюються повідомленнями і використовують погодженя в визначеному форматі. Компноненти системи виконують: відображення інформації, серверний додаток, менеджер ресурсів або сервер із базою даних. Особливіть полягає у тому, що відразу декілька серверів можуть опрацьовувати клієнтські запити, що веде за собою зниження навантаження на сервер. Зображення трирівневої архітектури показано на рисунку 2.2.

Рисунок 2.2 – Схематичне зображення трьохрівневої архітектури

Компноненти трирівневої архітектури виконують такі процеси:

– відображення даних клієнта;

– вирішує поставлені завдання серверний додаток, який виконує функції проміжного програмного продукту;

– керування ресурсами, сервер БД надає доступ до даних.

Трирівневу архітектуру можна перетворити на багаторівневу, тобто будуть встановлені додаткові сервери. Цим самим покращується ефективність системи, кожен сервер надає власні сервіси і може користуватися послугами інших підключених серверів.

Отже, двухрівнева архітектура простіша, усі запити опрацьовуються одним сервером, але вона менш надійна і покладає необхідність в використанні більш потужного серверу. Трирівневу архітектуру важче розробити і встановити, але отримуємо значний приріст в швидкодії, гнучкість системи і можливість її розширення, великий рівень надійності, для кожного серверу встановлюється свій захист, але викооистання мережевого трафіку зростає. Клієнт-серверна архітектура не ділить комп'ютери на клієнт і сервер, а дозволяє розподілити навантаження і функціонал між клієнтською і серверною частино.

Після опису принципів архітектурної роботи і побудови програмної системи, переходимо до опису шаблону проектування, який повинен інтегруватись у додатки для мобільного і настільного пристрою.

Шаблони проектування або патерни – це правила вирішення поставлених задач, це оптимізований спосіб вирішення проблеми. Патерн потрібно реалізувати, його не можна скопіювати, імпортувати в програму, це не конкретний код, а концепція для рішення завдань. Шаблон, потрібно підлаштувати під специфіку вашої програми, платформи і він не залежить від мови програмування. Реалізація відрізняється в мовах програмування.

Патерни відрізняються за складністю, деталізацією і обсягом системи, що розробляється.

Існують три основних типи шаблонів проектування:

– структурний;

– породжуючий;

– поведінковий.

Структурний патерн використовується для створення взаємодії між об’єктами, полегшуючи їх спільну роботу.

Породжуючий патерн полугшує створення об’єктів залежно від ситуації.

Поведінковий патерн потрібее для спілкування між об’єктами, роблячи їх більш простими і гнучкими.

Використання шаблонів проектування має багато переваг:

а) розробляючи програмну систему ми використовуємо провірені рішення і тим самим тратимо меншьне часу;

б) покращена стандартизація коду, що призводить до меньшої кількості непередбачуваних проблем, ми використовуємо типові рішення, в яких прорахували можливі несправності.

Якщо ми використовуємо слабку мову програмування, то у нас появляється необхідність в створені додаткової абстракції і патерн, є костилем.

Шаблон проектування MVC – виконує розділення даних системи на користувацький інтерфейс і модуль керування логікою, які діляться на окремі компоненти: модель, подання, контролер, це дає змогу змінювати і вдосконалювати окремі компоненти незалежно один від одного.

MVC, як і інші патерни, опису спосіб організації коду. Завдання зробити код більш простим у розуміні і щоб він бистріше оброблявся комп’ютером. Основна ідей, це розділення інтерфейсу користувача і логіки.

Компонент логіки додатку не повинен описувати зовнішній вигляд додатку, а подання немає опису логіки роботи. Головним елементом MVC, є контролер, він приймає запити користувача і вирішує, що з ними робити, взаємодіючи із моделю, яка обробляє інформацію і повертає її контролеру.

Після обробки інформації, контролер вирішує, що з нею робити. Контролер обов’язкова частина, інші компоненти необов’язкові.

Модель виконує стандартну роботу обробки даних і надає контролеру звіт, проводить обрахунки. Подання відображує заготовлені дані від моделі, які

перенаправляються контролером.

Отже, даний патерн чудово підходить для розробки веб-додатків і має певну схожість із клієнт-серверною архітектурою.

Шаблон MVVM дозволяє відокремити логіку програми від візуальної частини. Ця модель є архітектурною, тобто визначає загальну архітектуру програми. Шаблон має основні компоненти: Model, ViewModel і View.

Model – описує дані у додатку, містить логіку, яка пов’язана з цими даними. Модель не відображає дані та не взаємодіє з візуальними елементами управління.

Інтерфейси INotifyPropertyChangad та INotifyCollectionChanged, використовуються для інформування системи про оновлення параметрів моделі, які відбуваються. Це полегшує зв'язок із репрезентацією, хоча немає прямої взаємодії між моделлю та репрезентацією.

View – представлення подання, визначає візуальний інтерфейс, за допомогою якого користувачі взаємодіють із додатком. Подання - це код написаний на мові XAML, який відтворює інтерфейс.

Представлення не обробляє події, а виконує дії за допомогою команд. Однак іноді пов'язаний файл коду може містити логіку, яку важко реалізувати в шаблоні MVVM.

ViewModel – являється комунікативним механізмом між моделлю та поданням і ця комунікація відбувається за допомогою прив'язки даних. При змінні властивостей, інтерфейса INotifyPropertyChanged сигналізує про це, що змушує оновити дані у представлені.

ViewModel включає логіку отримання даних моделі, які потім відтворюються у представлені. Візуальні елементи взаємодіять через команди.

Результатом використання шаблону MVVM, є модульний розподіл програми на компоненти, які легше розробляти та перевіряти, а також в подальшому модифікувати та підтримувати.

2.2 Декомпозиції та детальне проектування

Детальне проектування, це наступна фаза розробки програмного забезпечення, яка починається після проектування архітектури і буде завершуватись на моменті отримання повного плану програмування. Проектування можна розбити на такі етапи:

- фіксування вимог;
- визначення класів;
- розробка архітектури;
- перевірка вимог.

Основна мета – підготовка проекту до його реалізації. Зв'язок між варіантами використання, архітектурою і проектуванням. Варіанти використання, це вимоги на основі яких, потрібно вибрати архітектуру і розробляти проект, який реалізує необхідні варіанти використання. Також на початкових етапах, особливу увагу потрібно приділити самим важливим архітектурним процесам.

Програмна система, буде розглядатися, як сукупністю об'єктів, які взаємодіють між собою, тобто об'єктно-орієнтована парадигма.

Вимоги до програмної системи, були визначені у попередніх розділах, зараз будуть описані класи предметної області, які відображають вимоги. Для детального проектування, буде описана модульна архітектура, проектування бази даних. Проектування функцій та опис інтерфейсів. Після чого проведеться вибір засобів реалізації проекту і підведення підсумків.

Розпочнемо декомпозицію і детальне проектування із опису бази даних. Проектування предметної області на етапі концептуального проектування, це модель сутність-зв’язок, її ще називають ER-моделю. Моделювання структури даних нашої предметної області базується на графічному або детальному описі і вони представляються зв’язок між сутностями.

База даних важлива складова програмної системи, що розробляється, основна її функція полягає в зберіганні даних про користувача, отримані даних сервером, і зміні відповідних дозволених атрибутів. Розберемо основні види таблиць бази даних для нашого проекту:

– User, користувач;

– WaitMessage, представлає спеціальну таблицю, яка матиме дані про вхідні повідомлення, які не були прочитані користувачем.

Для більш детального опису, використаємо логічну модель сутність-зв’язок, яка відображає концептуальну модель світу, об’ктами. Основними термінами у ній є:

– сутність, це об’єкт реального світу, а якщо точніше, вигляд об’єкту предметної області;

– атрибут, це опис сутності, її властивості;

– зв’язок або відношення, це асоціація, встановлена між кількома сутностями.

Розглянемо, які бувають види сутностей, їх атрибути і зв’язки. Бувають два види сутностей: сильні і слабкі, і вони діляться на типи і підтипи. Слабка сутність не може існувати без сильної сутності при рішення завдання. Підтип володіє додатковими атрибутами.

Реляційна модель даних заснована на моделі сутність-зв’язок, усі сутності із моделі відповідають відношеню, кожна властивість відповідає атрибуту відношення і у відношення додається лічильник, який є первинним ключом. Тепер розглянемо типи зв’язків.

Щоб спроектувати реляційну модель бази даних, потрібно розробити ER-модель, де буде розбиття відношень для формалізації даних.

На початку проектування бази даних, було описано декілька сутностей, опишемо їх. Дані зберігатються у зашифрованому форматі і сервер бере на себе навантаження по правильній обробці цих даних.

User – користувач програмної системи, який зареєструвався, звернення до цієї сутності буде відбуватися в моменти реєстрації та авторизації, атрибути:

– ID – номер користувача у базі даних, ключ;

– Login – представляє унікальне ім’я користувача;

– UserName – ім’я користувача, яке будуть бачити інші;

– Password – пароль;

– SeckerWord – спеціальне секретне слово відоме лише користувачу;

– WaitMessageID – посилання на таблицю із непрочитаними повідомленнями.

WaitMessage – таблиця повідомлень, які не прочитані, має атрибути:

– ID – ключ таблиці;

– Message – повідомлення, яке очікує відправлення;

– WhoSendLogin – логін користувача, який відправив повідомлення;

– WhoSendUserName – ім’я користувача, який відправив повідомлення.

Графічне відображення таблиць бази даних, відносно описаної вище структури, показано на рисунку 2.3.

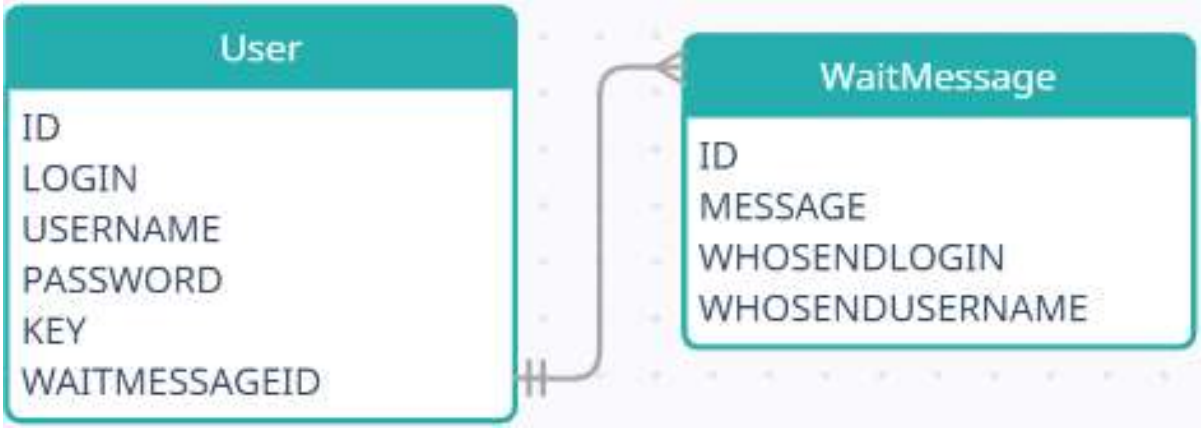


Рисунок 2.3 – Графічне представлення моделі сутність-зв’язок

Отже, була спроектована логічна модель бази даних, вибір системи керування базою даних проведеться при виборі засобів реалізації.

Після декомпозиції бази даних, переходимо до проектування користувацького інтерфейсу, а саме, способів взаємодії із ним. Опишемо логічні зв’язки і послідовності навігації між інтерфейсами.

Розробка користувацького інтерфейсу важлива частина, оскільки кінцевий споживач буде взаємодіяти із усіма можливостями додатку через розроблений інтерфейс. Він повинен бути простим і зрозумілим.

Інтерфейс повинен відповідати вимогам розписаним у першому розділі, оскільки ведеться розробка відразу трьох додатків: клієнтська програма на персональний комп'ютер, клієнтський додаток на мобільні пристрої та серверна програмна система на ком'ютер, для кожної із них потрібно розробити свій інтерфейс, але взаємодія у клієнтських додатків буде однакова, як для комп'ютера так і для телефону.

Отже, потрібно описати клієнтський інтерфейс взаємодії, потім серверну програму і описати взаємодію із нею, після чого розробити дизайн, який уже буде відрізнятися у різних клієнтів і у серверної частини. Користувацький інтерфейс буде складатися із вікон, сторінок, опишемо основні вікна користувача:

- форма авторизації;
- форма реєстрації;
- форма відновлення паролю;
- головне вікно у якому покзуються усі відкриті листування;
- вікно конкретного листування
- форма налаштувань.

Інтерфейс для серверної частини буде складатися в основному із вікон між якими буде можна переключатись:

- стан і нашаштування серверної частини;
- вікно із інформацією про нові підключення до серверу;
- вікно із інформацією про спроби авторизуватись;
- вікно із інформацією про спроби зареєструватись;
- вікно із інформацією про помилки;
- вікно із інформацією про активність клієнта.

Тепер розглянемо більш детально сторінки і вікна користувацького інтерфейсу і опишемо їх особливості.

Форма авторизації – початкове вікно, яке зустрічає користувача, містить дві форми для введення логіну та паролю та кнопку для авторизації, також два додаткових гіперпосилання на форми реєстрації та відновлення паролю.

Форма реєстрації – відкривається коли користувач хоче зареєструвати обліковий запису у системі, містить чотири поля для введення даних, а саме: логін, ім’я, пароль і підтвердження пароль. Засобом взаємодії виступає кнопка, яка повідомляє про спробу зареєструватися.

Форма відновлення паролю – відкривається коли користувач хоче відновити пароль до облікового запису, містить два поля для введення даних: логін та секретне слово. Кнопка на формі служить для спроби відновити пароль. Після натискання на кнопку, вас перенаправить на форму введення нового паролю, яка містить два поля для введення.

Форма налаштувань – містить інформацію про обліковий запис користувача, дозволяє змінювати деякі поля, перехід до форми відбувається по натисканю клавіші на боковому меню.

Головне вікно – містить список чатів, у вигляді прямокутних блоків, на кожному чаті видно ім’я користувача із яким ведеться листування та останнє повідомлення. В правому верхньому кутку розміщена клавіша для пошуку користувачів, кнопка для видалення чатів.

Вікно конкретного листування – перехід до вікна відбувається через натискання на головному вікні необхідного блоку чату, вікно містить усе листування із користувачем, повернутися до попереднього вікна, можна вибравши у боковому меню пункт «Головна».

Усі форми та вікна, крім авторизації, реєстрації та відновлення паролю, мають бокове меню справа. Вигляд меню та дизайн буде відрізнятися в залежності від платформи. На рисунку 2.4 показане графічне відображення взаємодії клієнтського інтерфейсу.

Тепер розглянемо сторінки та вікна серверної системи і опишемо їх особливості. Перехід ко кожного із вікон можна виконати вибравши відповідне вікно у верхньому меню додатку.

Рисунок 2.4 – Графічне зображення інтерфейсу клієнтського додатку

Сторінка налаштувань та стану серверу – перше вікно, яке зустрічає користувача, на ньому зображені основні налаштування серверу, поля для введення IP та порту сервера, якщо це необхідно, кнопки керування сервером: запуск, зупинка, отримання певної додаткової інформації.

Усі вікна будуть мати вигляд списку із датою та часом встановлених на сервері, крім сторінки із статусом та налаштуванням серверу. Зміст списку буде відовідати його назві, кожне вікно матиме кнопку для формування звіту по ньому та очистці даних.

Отже, був описаний користувацький інтерфейс із оглядом на його особливості та залежності.

2.3 Аналіз ти вибір технологій для реалізації програмної системи

Вибір мови програмування, важлива частина проектування програмного забезпечення, оскільки від її можливостей залежить: робота додатків, вибір технологій, важкість розробки.

Для реалізації даної програмної системи я вибрав мову програмування C#, на цій мові буде розроблений увесь необхідний функціонал.

C# – мова програмування, яка використовує об'єктно-орієнтований підхід до розробки. Мова має синтаксис близький до C++ та Java. Мова строго типізована, має велику кількість синтаксичного цукру, підтримує принципи ООП, а саме: поліморфізм; наслідування; інкапсуляцію.

Важливим фактором вибору мови Сі-шарп, є модульна платформа для розробки .NET Core. Даний фреймворк дозволить зробити додаток багатоплатформовим, що дасть змогу використовувати його на різних операційних системах.

Переваги мови програмування:

– підтримка продуктів Mircosoft;

– можливість створення програмних систем на мобільних і комп’ютрних платформах;

– велика кількість «синтаксичного цукру»;

– автоматична сбірка мусору, garbage collector;

– строга типізація, що полегшує розробку, типи даних мають фіксоване занчення;

– об'єктно орієнтоване програмування.

Мова чудово підходить і задовільняє наші потреби, які поставлені в процесі аналізу і проектування програмної системи.

Після вибору мови програмування, потрібно визначитися із технологіями, які підтримує ця мова. Технології повині задовільняти наші вимоги і добре працювати на вибраній мові. Мова програмування, є популярною і ефективною, вибір засобів реалізації можна зупинити на самих стабільних технологіях.

Для реалізації настільного додатка, я використовую технологію WPF – це аналог технології WinForms, яка працює лише на операційних системах Windows. Це підсистема для взаємодії графічних інтерфейсів.

Відмінність від старих WinForms полягає в тому, що для конструкцій традиційно використовувались сервіси, що базуються на WinForms, призначені для управління відтвореннями та для частин Windows, як User32 та GDI+, програми WPF базуються на DirectX, це важлива функція графічного

відображення в WPF. При використанні WPF велика частина роботи з відображенням графіки, як найпростіших кнопок, так і складних 3D-моделей, лежить на графічному процесорі на графічній карті, що також дозволяє використовувати прискорювач апаратна графіка.

До недоліків можна віднести:

– споживання великої кількості пам’яті, але це компенсується ширшими графічними можливостями та кращою продуктивністю графічного дисплея;

– додаток може працювати лише на операційній системі Windows.

Переваги WPF:

– розробка логіки виконується на мові програмування C#;

– усі версії операційної системи Windows підтримують цю технологію;

– щоб розробляти графічне вікно, використовується мова XAML, яка є альтернативою графічним та програмним елементам управління;

– поєднання XAML та C#

– у графічному вікні, є можливість масштабувати розмір додатку під різні екрани;

– набагато більше нових функцій;

– використання традиційних елементів керування WinForms;

– створення нових користувацьких контролерів;

– легкість відтворення анімації.

Мова програмування C# досить багата на велику кількість технологій, щоб реалізувати мобільний додаток, я використаю технологію Xamarin.Forms.

Xamarin призначений для створення додатків під операційні системи Android та iOS. Платформа вирішує проблему із якою стикаються розробники, при написані одного додатку, для різних операційних систем. Звичайно, щоб якісно створити програму систему, потрібно вносити корективи реалізації, які вимагають системи, але основний функціонал залишається тим самим.

Xamarin – це ефективний і простий інструмент для створення додатків під необхідні платформи одночасно. Це дозволяє мені створювати єдину логіку додатка за допомогою C#. Основні плюси використання Xamarin.Forms:

– непотрібно окремо розробляти додатки для різних операційних систем;
– кожна платформа надає свій API, через який проводиться розробка;
– для розробки додатків використовується мова програмування C#;
– розробити додатки, можна для декількох платформ.

У процесі проектування програмного забезпечення, потрібно застосувати протокол передачі даних через TCP/IP.

Замість використання стандартних доступних протоколів я вирішив написати власний протокол на рівні програми, верхній рівень моделі OSI, який забезпечує взаємодію з мережею та користувачами.

Отже, протокол передачі даних – це домовленість між програмами про те, як повинна виглядати передача даних.

Усі протоколи можна розділити на дві групи: символічні та двійкові або байтові. Бінарні протоколи, можна розділити на два шари: рівень контейнера та рівень даних.

Перший рівень, відповідає за цілісність і надійність передачі даних, а також за доступність розпізнавання повідомлень у байтовому потоці і звичайно, за зберігання повідомлень рівня даних.

Другий рівень, повинен містити інформацію, для якої розпочато всі взаємодії в мережі, у відповідному форматі для обробки. Їх структура в основному залежить від завдань, які потрібно вирішити, але для цього існують загальні рекомендації.

Перетворюючи відповідь за відомим зразком, програма надає інформацію користувачеві. Здійснити аналіз цього пакету можна лише маючи інформацію про його структуру.

Розроблюваний протокол буде бінарним і матиме назву EMTP (Easy Message Transfer Protocol). Протокол як і додатки розробляється на мові програмування Сі-шарп.

Як було описано, безпеку протоколу забезпечить алгоритм шифрування AES, який входитиме стандартом в структуру протоколу і матиме елементи зміни

ключів для різних систем. Структура спроектованого протоколу передачі даних у вигляді пакету даних показана на рисунку 2.3.

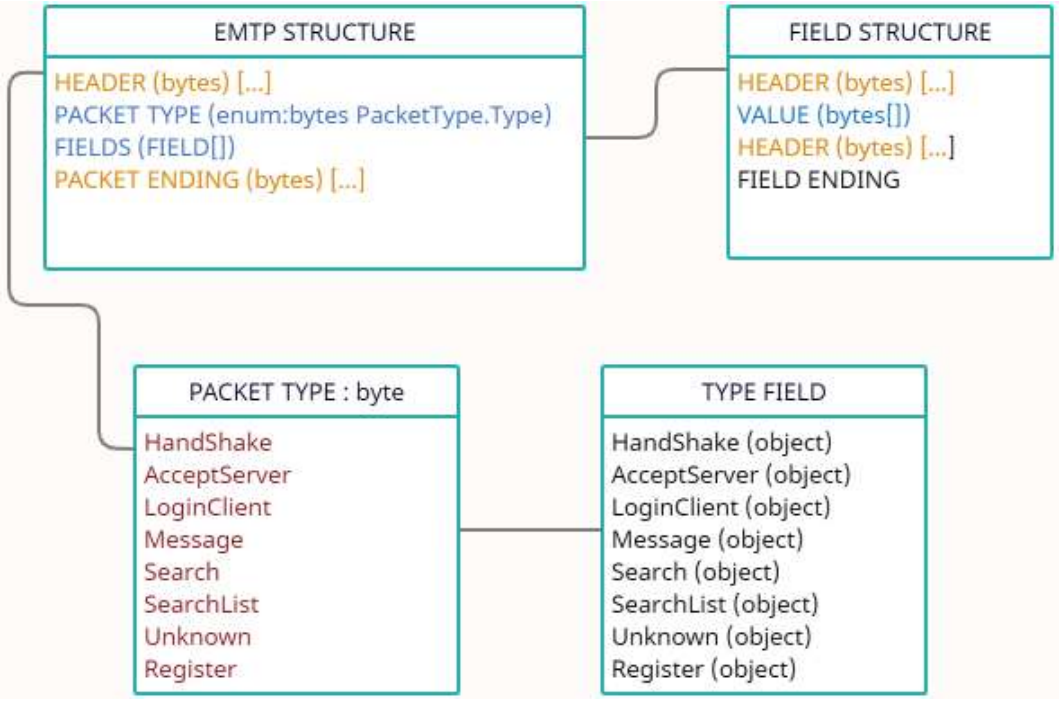


Рисунок 2.3 – Структура EMTP-пакету

База даних – зберігає у собі інформацію про об’єкти. Існує велика кількість типів баз даних, але основними із них є три: ієрархічна база даних, мережева база даних, реляційні бази даних.

Ієрархічна база даних – об’єкти в такому виді зберігаються, як певний зміст, можлива наявність дочірніх елементів, батьківських елементів. Дочірні можуть мати ще дочірні елементи і для них бути батьківськими. Така база даних чудово підходить для зчитування даних, бистрий вибір і відправка даних, але покладаються обмеження перебирання інформацією.

Мережева база даних – модифікація ієрархічного типу. Елементи в мережевій базі даних, можуть мати довільну кількість батьківських елементів, а ієрархічні тільки один. Характеристики майже аналогічні ієрархічній базі даних.

Реляційні бази даних – об’єкти всередині цього типу бази даних зберігаються у вигляді двухрівневої таблиці, яка складається із стовбців в яких виводяться назви і опис даних, кількість стовбців завжди фіксована і задається на

етапі створення БД, а заповнювати їх можна довільною кількістю даних або поки вистачає місця.

Щоб задовільнити вимоги і ефективно інтегрувати систему керування базами даних у нашу програмну систему, потрібно використовувати MySQL.

Забезпечити безпечний спосіб обміну даними, це важлива складова програмної системи, що розробляється, оскільки буде використовуватись протокол передачі даних, який передаватиме бінарний масив череж мережу, його потрібно захистити алгоритмом шифрування.

Алгоритми шифрування існують двох видів: симетричні і асиметричні. Вони повинні відповідати криптографічній стійкості, тобто алгоритм повинен протидіяти криптоаналізу. Стійкими вважаються алгоритми, який для взлому вимагає недосяжної продуктивності від того хто намагається його взломати або потребує значних затрат у часі на взлом. Криптостійкість не можна математично довести, доводиться лише вразливість алгоритму.

Умови, які повинна задовільняти криптосистема, це стійкість, яка залежить від можливості читання даних. Абсолютно стійкі криптосестеми, це системи, які неможливо взломати теоретично і практично. Вимоги для абсолютної стійкості:

– генерація ключа виконується для кожного обміну повідомленнями;
– надійність ключа;
– довжина ключа відповідає довжині повідомлення;
– повідомлення має надлишок тексту.

Достатньо стійкі системи, це коли криптосистама використовує практичну стійкість або вирахункову, тобто взломати систему можна при умові отримання ключа шифрування, але зараз немає досить потужних систем для взлому. Розглянемо алгоритми шифрування.

Симетричні алгоритми використовують ключ, який використовується для шифрування і дешифрування тексту, ключ завжди той самий. Вони діляться на поточні і блочні, перший об’єднується із потоком випадкових цифр шифру, другий беручи кількість байтів шифрує їх, як одне ціле. Відправник і отримувач повинні мати копії одного ключа і зберігати їх у місці, де ніхто не зможе отримати

до нього доступ. Недоліком являється використання одного і того ж ключа, оскільки отримавши ключ, можна розшифрувати усе повідомлення.

Перерахуємо деякі симетричні алгоритми:

– AES, використовується і по цей час, розширений стандарт шифруваня даних;

– DES, стандарт шифрування, зараз вважаються небезпечними;

– MARS, підтримує 128-бітні блоки, надає сильну протидію криптографічним атакам;

– IDEA, зараз широко не використовується, був заміною DES.

Асиметричні алгоритми шифрування, це криптографія із публічним ключем, принцип, який використовується, передбачає наявність пари зв’язаних ключів, які виконують шифруванням і дешифруванням. Відкритий ключ, може отримати кожен, але закритий ключ зберігається в таємниці. Кожен має змогу зашифрувати повідомлення маючи відкритий ключ, але лише той хто володіє секретним ключом, може розшифрувати його.

Перерахуємо деякі асиметричні алгоритми:

– RSA, зараз найбільш використовувана асиметрична криптосистема, алгоритм заснований на факторизації простих цифр. Проблема полягає у тому, що алгоритм повільний;

– ECC, криптографія із еліптичними кривими, криптографічні алгоритми зазвичай використовують математичне рівняння для розшифровки ключів;

– Diffie-Holman, зараз цей алгоритм не є стандартом, оскільки були виявлені вразливості.

Отже, проаналізувавши криптографічну стійкість, алгоритми криптографічних систем для шифрування даних і вимоги до програмної системи, робимо висновок, що нам найбільше підходить алгоритм шифрування AES.

Отже, був проведений аналіз і порівняня клієнт-серверних архітектур і розібрані способи проектуання таких систем. Також були описані і порівняні патерни проектування програмних систем. Проведений опис користувацького інтерфейсу, який відповідає вимогам системи.

Результатом аналізу визначено, що програмний продукт буде розроблятися на клієнт-серверній архітектурі, створення системи ведеться в двухрівневому стилі із товстим клієнтом і сильним сервером. Додатки будуть розроблятися на шаблоні проектування MVVM.

Сховище даних, важлива частина програмної сисетми, що розрорбляється. Для керування, зберігання даних, була обрана реляційна система MySQL, яка забезпечить безпкеку даних.

Вибрані технології реалізації клієнтських додатків, що відповідають вимогам до програмної системи і доповнюють вибрані архітектурні рішення.

Описаний спосіб мережевої взаємодії через протокол, виникла необхідність у створені власного прикладного протоколу передачі даних. Проаналізована криптостійкість протоколів і алгоритмів, в результати вибраний стійкий алгоритм шифрування для протоколу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Детальне проектування модулів

Проводячи проектування модулів програмної системи, будуть описані способи роботи модулів і способи взаємодії із: протоколом та асинхроним отриманням і відправкою даних. Розпишеться загальна структура і особливості архітектури, способи взаємодії між модулями. Будуть приведені діаграми класів, якщо це буде необхідно. Розпочнемо із проектування протоколу передачі даних.

Протокол важлива частина програмної система, що розробляється. З його допомогою клієнт і сервер будуть обмінюватись інформацією один з одним. Процес спілкування буде виконуватись шляхом відправки зашифрованих пакетів із даними. Клієнт або сервер на початку повинні створити пакет і потім його відправити. Коли вони отримують зашифрований пакет його необхідно дешифрувати і розібрати на складові протоколу.

Отже, інтерфейс взаємодії у коді із протоколом передачі даних, буде виконуватись такими функціями:

– Create: створює пакет (потрібно вказати тип пакету, після чого передати в створений пакет дані);

– Parse: робить розбір вхідних даних і перетворює їх в пакет відповідного типу, викликаючи відповідні методи всередині функції, які виконують перетворення дочірнього класу у батьківський;

– ToPacket: перетворює пакет у зашифрований потік даних;

– SetValue: додання у пакет даних, які відповідатимуть його типу;

– GetValue: отримання даних розібраного пакету;

Типи пакетів повинні відповідати завданню, яке вони виконують. Функції отримання і задання даних будуть відображати тип пакета. Тепер перерахуємо потрібні типи пакетів і коротко опишемо їх:

– HandShake: тип пакету, який потрібен для рукостискання між клієнтом та сервером, що буде повідомляти підключення клієнта до серверу;

– AcceptServer: тип пакету, який відправляється лише клієнту і є відповідю на спробу зареєструватися або авторизуватись;

– LoginClient: тип пакету, який надсилає клієнт із даними для авторизації в системі;

– Message: тип пакету, який надсилає, як клієнт так і сервер, і призначений для передачі повідомлення між клієнтами;

– Search: тип пакету, який надсилає клієнт для отримання списку користувачів, яких він шукає, щоб почати спілкування;

– SearchList: у пакеті знаходиться список користувачів;

– Unknown: тип пакету, який повідомляє про помилку відправки або парсингу вхідного пакету;

– UserContent: тип пакету, який потрібен, щоб користувач отримав свої дані, а сервер встановив взаємодію із клієнтом;

– Register: тип пакету, який відправляє клієнт із даними для реєстрації, а сервер обробляє два сценарії після отримання.

Щоб коректно отримувати дані при обробці вхідного пакету, кожному типу буде відповідати клас із приставкою Field, тому після отримання пакету і коректної обробки, потрібно скористатись функцією GetValues, яка поверне об’єкт. Його потрібно привести до класу із певними полями, це означає, що на руки додатків, а не протоколу лягає завдання створення менеджера отриманих пакетів, який опрацьовуватиме вхідні дані.

Оскільки, ми можемо передавати дані по мережі у вигляді масиву байтів, перед відправкою пакету, його потрібно перетворити у потік байтів, для цього використовується функція ToPacket, яка перетворює об’єкт пакету у масив байт і шифрує його алгоритмом AES.

Тепер розглянемо типові способи взаємодії із протоколом, на прикладі створення і відправки пакету із типом HandShake. На рисунку 3.1 графічно представлений процес створення і відправки пакету, а на рисунку 3.2 показаний процес отримання пакету.

Рисунок 3.1 – Процес створення і відправки пакету

Рисунок 3.2 – Процес отримання пакету

Отже, кожен спосіб взаємодію проходить етап створення або отримання об’єкту із полями пакету або його успадкування від базового класу, увесь процес: шифрування, дешифрування, розпізнавання пакету, відбувається в середині протоколу, він надає методи для роботи із ним Типова структура пакету була описана у попередньому розділі, а тепер подивимося на діаграму класів, яка показана на рисунку 3.3.

Дивлячись на діаграму, робимо висновок, що усі властивості пакетів будуть надаватись у відповідному типові даних, який унікальний для кожного пакету, усю роботу по перетворені даних у масив байт і навпаки бере на себе протокол, щоб взаємодіяти із пакетому досту

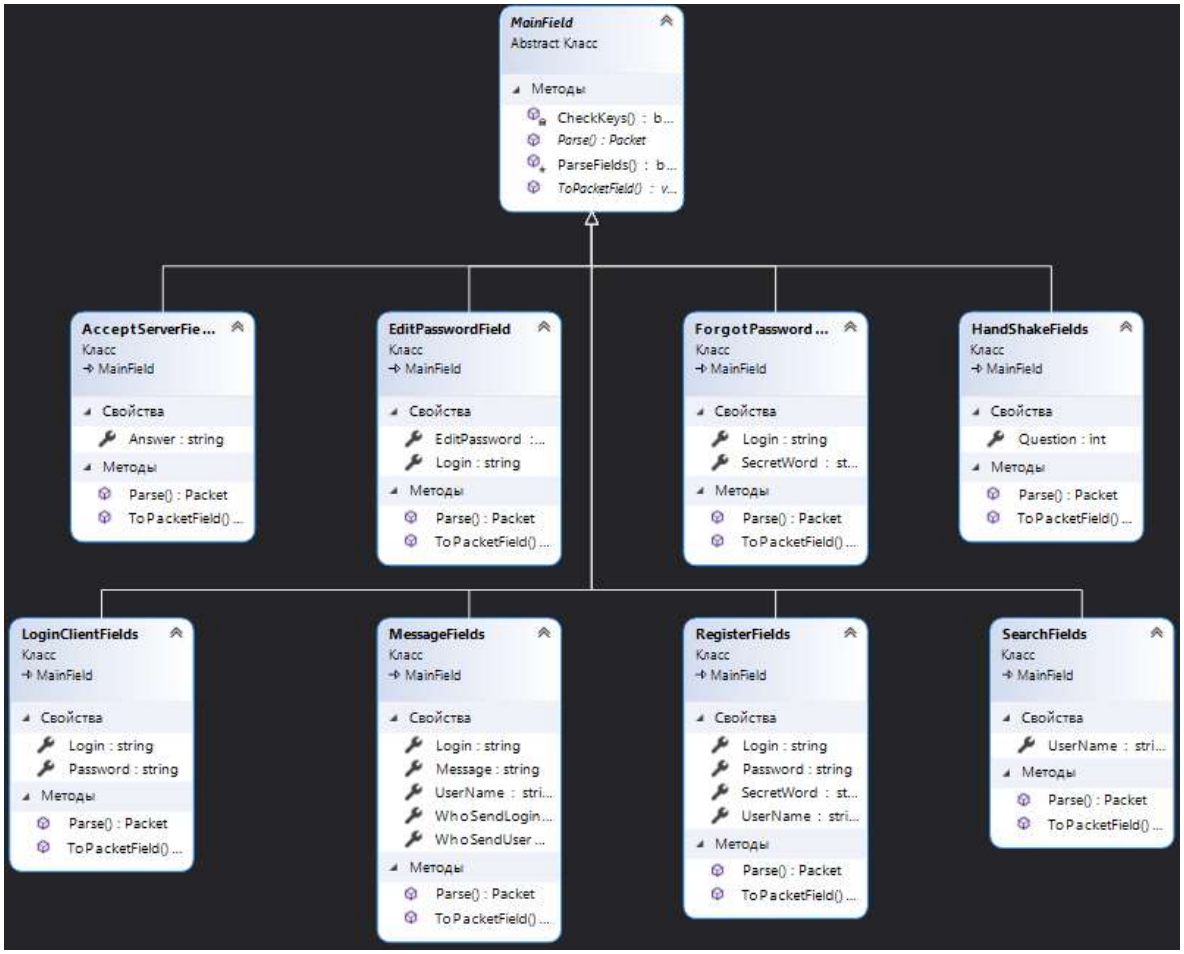


Рисунок 3.3 – Діаграма класів полів пакету

Тепер розглянемо клас, який буде відповідати за шифрування нашого пакету. Як було сказано раніше, алгоритм шифрування був вибраний AES, відповідно потрібен клас, який буде реалізовувати даний алгоритм. Оскільки, алгоритм нам потрібен при перетворені пакету у масив байт і коли ми перетврюємо масив байт у пакет, цей клас буде використовуватись у методах ToPacket і Parse. На рисунку Б.1 можна подивитися на діаграму класу пакету.

Структура класу, який проводить шифрування і дешифрування даних представлена на рисунку 3.4.

Секретне слово, яке потрібно для роботи алгоритму та додаткові поля, які приймають участь в роботі класу, прописані константою, змінюючи їх програма продовжить працювати коректно. Щоб комфортно працювати із класом, можна створити глобальну зміну для його екземпляра.

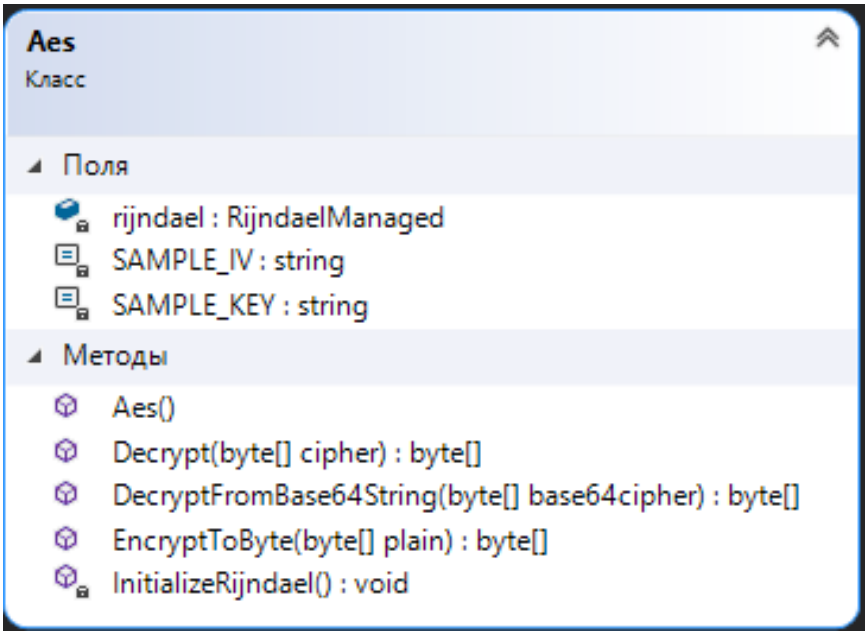


Рисунок 3.4 – Структура класу шифрування пакету

Переходимо до проектування модуля, який відповідах за асинхрону роботу додатку, що включає, в основному обмін даними із сервером.

Додатки повині працювати в асинхроному режимі, тобто більше ніж в 1 потоці, це дасть можливість обробляти велику кількість вхідних даних від серверу і відправляти дані, не блокуючи основне вікно програми. Також є сенс у виділені необхідних для взаємодії властивостей клієнту і створені загального об’єкту, який назвемо Client. Для коректної роботи модуля потрібно виділити його властивості, опишемо їх:

– Buffer: являється масивом байт, в нього буде записуватись отриманий протокол. Оновлення його потрібно робити окремо;

– TcpClient: потрібно для підключення клієнта до мережевого з’єдання по TCP і для отримання потоку даних мережі;

– NetworkStream: потік даних у мережі, який отримуємо з TcpClient, також потрібен для отримання даних і відправки;

– EventWaitHandle: необхідна подія, яка буде синхронізувати потік, використовується при отриманні повідомлення;

– Task: використовується для асинхроної операції передачі повідомлення.

Оскільки деякі властивості класу потрібно використовувати у додатках, а не тільки у реалізації роботи обміну даними, визначимо доступні поля для взаємодії. Нам потрібні поля: Buffer, TcpClient і NetworkStream, інші використовуються всередині класу.

Тепер опишемо методи, які виконують необхідний функціонал:

– Send: отримує в якості вхідного параметру масив байт, який являється пакетом і відправляє його, використовує поле Task;

– BeginRead: починає асинхроне читання даних, очікує потік байтів не навантажуючи систему, після чого заповнює Buffer вхідними даними.

Отже, ми описали поля нашого модуля і доступні методи із якими відбуватимется взаємодія у коді, такі методи, як: EndRead, BeginWrite і EndWrite являються приватними і потрібні для асинхроного отримання і відправки даних, їх програмна реалізації буде описана далі.

Цей клас матиме усі необхідні поля і методи для мережевої взаємодії клієнта і серверу. Діаграма класу цього модуля представлена на рисунку Б.2. Сервер використовуватиме цей клас для ідентифікації клієнта і роботи із цими полями, це полегшить роботу із великою кількістю клієнтів.

Розпочинаємо проектування WPF, додатку і реалізації шаблону проектування. У попередньому розділі було описано, що для додатків буде використовуватись шаблон проектування MVVM. Оскільки, клієнтський додаток для настільного пристрою і серверний додаток будуть розроблятися на WPF, то і реалізація шаблону і спосіб побудови додатку буде однаковий.

Отже, для реалізації шаблону проектування, нам потрібно розділити структуру проекту на такі складові: Models, Views, ViewModels, Infrastructure - це основні складові для коректної роботи додатків. Також будуть потрібні додаткові модулі, вони являються допоміжними і необхідними для реалізації тих чи інших функцій.

Розберемо основний сполучний компонент шаблону – ViewModel. Він повинен реалізовувати інтерфейс INotifyPropertyChanged, який потрібен для слідкування за змінами даних. Оскільки, для кожного вікна сторінки знадобиться

своя модель представлення, потрібно створити базовий батьківський клас, який об’єкти моделей представлення будуть наслідувати, що позбавить від копіювання коду і зайвої реалізації.

Розглянемо код базового ViewModel, який представлений у лістингу В.1. Кожне вікно або сторінку потрібно прив’язати до відповідної моделі представлення. Клас реалізовує інтерфейс і описаний метод OnPropertyChanged, також метод, який потрібен для заміни даних у полях класів. Розглянемо приклад створення поля:

```
private string _login;

public string Login
{
  set => return _login;
  get => Set(ref _login, value, nameof(Login));
}
```

Приєднання поля до відповідного візуального елементу у представленні показано нижче:

```
<TextBlock
HorizontalAlignment="Center"
Text="{Binding Path=Login}"/>
```

Тепер кожна ViewModel повинна унаслідувати даний клас. Також, потрібно реалізовувати створення конструкторів класу для моделей представлення. Таким чином, ми можемо взаємодіяти із представленням WPF через Binding.

Оскільки, при розробці додатку використовуючи шаблон MVVM потрібно максимально зменшити написання коду у файлі представлення, необхідно вирішити проблему взаємодії користувача із додатком. Раніше для цього використовувались події, які прийшли від WinForms, але вони вимагають створення методів подій у файлах самих представлень, нам потрібно використовувати конструкцію Binding і для елементів керування передавати методи у атрибут Command.

Отже, у папці Infrastructure ствоюється клас, який буде обробляти команди, які замінять нам події. Для цього потрібно реалізувати інтерфейс ICommand, який знаходиться у просторі імен System.Windows.Input, лістинг коду базового класу команди наведений у лістингу В.2. Важливим компонентом є використання CommandManager для реєстрування команди. Код приведений нижче:

```
CommandExecute
{
    CommandManager.RequerySuggested++;
    CommandManager.RequerySuggested--;
}
```

Тепер потрібно розробити клас, яким проводитиметься реєстрація команд, ми будемо передавати методи, які повині виконуватися. Реалізація приведена у лістингу В.3. Розглянемо приклад створення і реєстрації команди у ViewModel:

```
internal ICommand Command { get; private set; }
private void TestCommandExecute(object obj)
{
    Method();
}

private bool CanTestCommandExecute(object obj)
{
    return true;
}

public MainWindowViewModel()
{
    TestCommand = new RelayCommand(TestCommandExecute,CanTestCommandExecute);
}
```

Після створення необхідних методів, реалізацію команди обов’язково потрібно прописати у конструкторі класу і після чого можна використовувати у представлені. Приклад використання команди у представлені приведений нижче і показаний кодом XAML:

```
<Button
Grid.Column="2"
Content="Login"
Command="{Binding Path=TestCommand}"
Style="{StaticResource primarybutton}"/>
```

Приклад взаємодії класів моделі представлення із базовим класом показаний на рисунку Б.3, усі подібні класи матимуть схожу структуру і будуть відрізнятися полями.

Отже, ми розглянули проектний спосіб роботи додатку на шаблоні MVVM і описали способи взаємодії із даними. Тепер проведемо детальне проектування компонентів додатку, які потрібні для вирішення поставленої проблеми.

Для початку, розберемо способи авторизації у обліковому записі. Нам потрібні форми, щоб виконати вхід у акаунт, форма для реєстрації, відновлення паролю та введення секретного слова. Зараз наведемо основні елементи керування, які будуть розміщені на формах:

– форма входу в акаунта, матиме два поля для ведення даних, кнопку для виконання операції і дві кнопки для переходу до форм реєстрації та авторизації;

– форма реєстрації на ній знаходяться поля для введення усіх необхідних даних і кнопка для продовження реєстрації;

– форма відновлення паролю, схожа із формою входу в акаунт, але виконує трохи іншу функцію;

– форми введення секретного слова і введення нового паролю матимуть просто поля для введення даних, кнопку для підтвердження і кнопку відміни операції.

Усі ці форми будуть виконуватись незалежно одна від одної, за виключенням передачі певних даних між моделями представлення. Кожен раз буде відбуватись підключення до серверу.

Тепер, розглянемо основну форму, на якій знаходитимуться елементи користувацького інтерфейсу, які виконуватимуть функції: відображення списку доступних чатів, вікно із конкретним чатом в якому ведеться спілкування користувачів, вікно для пошуку користувачів і кнопка налаштувань облікового запису, яка активна завжди.

Основна форма буде знаходитись у одному вікні і усі вище описані функції будуть розміщені на ній, тобто, усе керування проходитеме в одній моделі представлення. Бокове меню матиме дві кнопки: налаштування акаунту та вихід

із акаунту. Менше половини основної форми займатиме список чатів. Інша частина форми буде виділена під чат і якщо чат не буде вибраний, користувач бачитиме пустий лист, також над чатому буде розміщена кнопка для закриття цього чату або його видалення.

Щоб знайти користувача, потрібно буде ввести його ім’я у відповідному полі і натиснути клавішу пошуку, якщо ім’я було введене коректно, появиться нова кнопка, яка повідомлятиме користувача, що пошук був вдалий і надасть можливість розпочати листування.

Тепер перейдемо до проектування мобільного додатку, який буде розроблятись використовуючи шаблон проектування MVVM.

Додаток для мобільного пристрою, буде розроблятися на мові C#, і також, при розробці використовуватиметься шаблон проектування MVVM. Хоча підхід до розробки дотатків однаковий, але потрібно брати до уваги особливості реалізації шаблону і платформи.

Аналогічно WPF додатку у нас є базові складові структури проекти. Реалізація базового класу однакова і наведена у лістингу В.1, наслідування і використання збігаються із прикладом приведеним у описі WPF додатку.

Оскільки, в проектах типу Xamarin відсутній клас CommandManager, реалізацію команд, як у WPF виконати неможливо, для цього потрібно використати клас Command із простору імен Xamarin.Forms. Лістинг коду реалізації класу для виконання команд приведений у лістингу В.4.

Тепер, після реалізації класу, який можна використовувати у якості команд або подій для візуальних інтерфейсів, розглянемо приклад створення команди у модулі представлення:

```
public Command AddCase
{
  get => return new RelayCommand(OnAddCaseExecute, CanAddCaseExecute, this);
}

private void OnAddCaseExecute(object obj)
{
  Method();
}
```

```
private bool CanAddCaseExecute(object obj)
{
  return true;
}
```

Прив’язка моделі представлення до представлення відбувається аналогічним чином, вказуючи у DataContext сторінки відповідний шлях до класу ViewModel, який реалізує базовий клас і інтерфейс INotifyPropertyChanged.

Тепер перейдемо до проектування компонентів додатку і визначимо способи вирішення поставленого завдання. Фактично структура сторінок і їх зміст будуть відповідати додатку на WPF, різниця буде у способі реалізації через відмінність платформ, що також відобразиться на проектуванні додатку.

Розподілимо додаток на дві частини, а саме – частина, яка відповідає за авторизацію користувачів, сюда входять:

– реєстрація;
– відновлення паролю;
– авторизація.

Та частина, яка відповідає за основний функціонал додатку, тобто спілкування між користувачами шляхом обміну повідомленнями, сюда входять:

– перегляд активних чатів;
– пошук користувачів;
– чат із конкретним користувачем;
– дані про обліковий запис.

Отже, для комфортної навігації між цими сторінками було вирішено використовувати елементи, що спливають, як навігаційне меню додатку. Щоб створити зв’язок між цими сторінками і побудувати надійну навігацію, потрібно в оболонці Xamarin.Forms використати елемент візуального інтерфейсу, як Shell, з його допомогою буде створена ієрархія сторінок.

Відповідно до проектування візуального інтерфейсу в мобільному додатку буде вспливаюче навігаційне меню, але воно потрібно лише для другої частини додатку, яка відповідає за обмін повідомленями, а у першій, просто виконуватиме

функцію навігації. Тепер визначимо усі необхідні сторінки, які будуть у додатку.

Для частини із авторизацією користувача потрібно:

– сторінка, яка матиме поля для введення даних, щоб провести авторизацію, відповідно кнопка, яка виконуватиме цю операцію, також навігаційні елементи, які будуть направляти користувача до сторінки реєстрації і відновлення паролю;

– сторінка реєстрації, на ній знаходяться поля для введення даних і кнопка, яка виконуватиме реєстрацію, якщо спроба буде успішною, користувача направить на строрінку введення секретного слова;

– сторінка відновлення паролю потрібна, щоб відновити пароль в уже існуючому обліковому запису, на ній знаходяться поля для введення логіну і секретного слова та кнопка виконання операції. Якщо дані введені коректно, то користувача направляють на сторінку введення нового паролю;

– сторінка введення секретного слова, матиме поле для введення секретного слова і дві кнопки для підтвердження реєстрації, кнопку відміни реєстраці, та кнопку налаштувань;

– сторінка для введення нового паролю, матиме два поля для введення нового паролю і підтвердження його, а також дві кнопки для зміни паролю, кнопку відміни операції.

Вище були описані сторінки для авторизації користувача. Тепер опишемо сторінки, які відповідають за обмін повідомленнями:

– сторінка на якій знаходяться усі активні чати у вигляді списку. Якщо натиснути на якийсь із чатів, відразу відкриється сторінка із детальним виглядом цього чату, де і проводиться спілкування між користувачами;

– сторінка налаштуваня облікового запису, на ній знаходиться необхідна відкрита інформація про обліковий запис і кнопки із назвами, які символізують редагування акаунта і які направлятимуть користувача на сторінки, де можна провести редагування відповідного поля;

– сторінка конкретного листування, ця сторінка виконуватиме функцію чату, на ній будуть зображені усі повідомлення, які були відправлені і отримані

у ході спілкування. Сторінка матиме форму для введення повідомлення і кнопку для його відправки, щоб повернутися до списку чатів потрібно скористатись боковим навігаційним меню.

У цій частині кожна сторінка матиме бокове навігаційне меню для простоти переходу між сторінками, також у меню буде знаходитись кнопка для виходу із облікового запису.

Отже, були описані сторінки мобільного додатку із вказанням необхідних візуальних елементів для коректної роботи. Тепер перейдемо до деталей реалізації додатку.

Для того, щоб передавати необхідні файли у додатку, які можуть бути потрібні у різних частинах моделей представлення, прийнято рішення створити клас Content, який буде реалізований патерном singleton, у ньому будуть знаходитись дані для авторизації і клас Client, який виконує функції асинхроного отримання і передачі даних.

Як вже було сказано, для навігації буде використовуватись візуальний елемент Shell, а кожна сторінка відповідатиме візуальному елементу ContentPage до якої підключатиметься модель представлення.

Отже, після проведення проектування мобілного додатку і опису важливих проектних рішень, переходимо до конкретної реалізації.

3.2 Програмна реалізація модулів

Після того, як було спроектовано протокол передачі даних, можна приступити до програмної релазіації цього компоненту додатку. Як вже було сказано, увесь програмний продукт розробляється на мові програмування C# і буде використовуватись .NET Core.

Отже, протокол представлятиме на виході бібліотеку із розширенням .dll. На початку створимо головний клас пакету, який називається Packet, простір імен

буде використовуватись, як назва протоколу, тобто EMTP. Тепер опишемо конструктор цього класу у лістингу нижче:

```
public EMTProtocolPacket Type { get; set; }
public static EMTPacket CreateEMTPacket(PacketType type)
{
  return new EMTPacket()
  { Type = type};
}
```

Під час звернення до пакету, ми не зможемо створити його, як звичайний клас, потрібно використати статичний метод Create, який поверне екземпляр цього класу. В конструктор потрібно передати тип пакету. Приклад ініціалізації пакету приведений у лістингу нижче:

```
var acceptPacket = Packet.Create(PacketType.AcceptServer);
```

Після того, як ми створили тип пакету потрібно передати у нього клас із відповідними полями, для цього використовуємо метод SetFields, який в якості параметра отримає клас типу MainField. Цей клас являється абстрактним і потрібен, щоб вказати тип кожному класу, який матиме відношення до полів пакету, а також цей клас потрібно реалізувати кожному хто його наслідує. Є два методи, які відповідають за формування пакету, які використовуються усімя нащадками. Лістинг коду представлений нижче:

```
public abstract class MainField
    {
        public abstract void ToPacketField(List<byte> packet);

        public abstract Packet Parse(byte[] field, Packet packet);

        protected byte[] ParseFields(byte[] packet, byte[] keys)
        {
            int count = 0;
            for (int i = 0; i < packet.Length; i++)
            {
                if (CheckKeys(packet, keys, i))
                    count = i;
            }

            return packet.Take(count).ToArray();
        }
```

```
        private bool CheckKeys(byte[] packet, byte[] keys, int count)
        {
            int goodCheckCount = 0;
            for (int i = 0; i < keys.Length; i++, count++)
            {
                if (packet[count] == keys[i])
                    goodCheckCount++;
            }

            return goodCheckCount == keys.Length;
        }
    }
```

Два додаткових методи потрібні для методу Parse, який реалізовується у класах, які наслідують його. Приклад заповнення полів приведений далі:

```
acceptPacket.SetFields(new AcceptServerFields() { Answer = "SERCERWORD" });
```

Тепер розглянемо, як виглядатиме клас на прикладі AcceptServerFields:

```
public class HandShakeFields : MainField
{
    public int Question { get; set; }
    public override void ToPacketField(List<byte> packet)
    {
        var numberByteArra = Packet.FixedObjectToByteArray(Question);
        Packet.Set(packet, numberByteArra);
    }

    public override Packet Parse(byte[] field, Packet packet)
    {
        var question = Packet.ByteArrayToFixedObject<int>(field);
        packet.SetFields(new HandShakeFields() { Question = question });
        return packet;
    }
}
```

Аналогічним чином будуть побудовані усі інші класи, які мають поля для певного типу пакета.

Реалізація методу SetFields досить проста, нам просто потрібно присвоїти полю PacketField клас, який передається у метод. Реалізація приведена нижче:

```
public void SetFields(MainField field)
{
    if(field != null)
        PacketField = field;
}
```

Після того, як буде отриманий пакет і розшифрований, потрібно отримати від нього дані, для цього використовується метод GetField, який повертає клас MainField, після чого його потрібно привести до відповідного типу і отримати дані. Реалізація методу приведена нижче:

```
public MainField GetFields()
{
    if(PacketField != null)
    {
        return PacketField;
    }

    return null;
}
```

Ми маємо можливість передавати дані у пакет і отримувати їх, що зроблено, як наслідування класів. Тепер необхідно реалізувати головні методи пакету, а саме, приведення класу до пакету ToPacket і методу, який буде перетворювати вхідний потік байтів у наш пакет Parse.

Розпочнемо із методу ToPacket, він повинен повератити масив байтів, який уже буде зашифрований і приведений до структури пакету, яка була описана у попередньому розділі.

Отже, у цьому методі потрібно на початку вказати заголовок пакету і в кінці, після заповнення полей пакету, записати його кінець і зашифрувати масив байт. Для початку, в якості контейнера для байтів будем використовувати List<byte>, оскільки в ньому легко змінювати кількість елементів, які туда входять. Потрібно написати простий метод, який буде заповнювати цей лист даними, його реалізація приведена у лістингу В.5. Сама реалізація методу також наведена у лістингу В.6.

Метод створює екземпляр класу, який відповідає за шифрування пакету і в конструкції return викликає метод, повертаючи зашифрований масив байт, який можна використовувати у інших методах.

Тепер, розглянемо метод Parse, його завдання полягає в розпізнавані пакету, відповідно у ньому потрібно дешифрувати масив байт і провести аналіз

отриманих даних на відповідність структурі пакету, і у випадку не відповідності повернути, як пустий пакет, реалізація у лістингу В.7.

Спочатку проводиться дешифрування вхідних даних і створення класу шифрування, потім виконується аналіз заголовків розшифрованого пакету і якщо валідація даних пройшла успішно, продовжується формування пакету із отриманих даних і результатом ми отримуємо готовий пакет даних.

Варто сказати, що для кожного типу пакета проводиться індивідуальне перетворення у пакет і розбір вхідних даних, оскільки різні типи мають різну кількість параметрів, які потрібно правильно вписати у структуру пакету.

Також використовуються допоміжні методи, які призначенні для перетворення байтів у строкове представлення і спеціальні методи, які перетворюють структурні типи даних у байтовий масив.

Цей метод перетворює зміну *int* у масив байт, який записується в структуру масиву і передається по мережі.

Отже, була описана реалізація важливого модуля програмної системи. Протокол буде використовуватись при взаємодіє клієнта та сервера. Зручність передачі даних, їх отримання і саме головне безпека, це завдання, які виконує даний протокол і чудово із ними справляється.

Переходимо до розробки модуля комунікації і асинхроної взаємодії із даними додатку, який використовуватиметься усією системою.

Даний модуль відіграє важливу роль, він буде використовуватись, як клієнтом так і сервером, і головна його мета, це забезпечити асинхроний обмін даними без блокування користувацького інтерфейсу і надати комфортний у використані набір полів, які можна легко скласти у ім’я Client. Для роботи із модулем потрібно буде лише екземпляр цього класу, і можна читати і відправляти повідомлення із масиву байтів.

Це усі необхідні поля, які будуть використовуватись. Подія EventWaitHandle, буде ініціалізовуватись у конструкторі класу, при створені класу, потрібно заповнити усі ці поля. Приклад заповнення полів приведений у коді нижче:

```
pc = new PC();
pc.Client = new TcpClient();
pc.Netword = obj.TcpClient.GetStream();
pc.Data = new byte[65536];
```

Також у класі знаходяться методи для читання та відправки даних із транспортного протоколу. Метод BeginRead починає зчитувати дані і використовує подію WaitHandle, щоб повідомляти коли потік отримав дані і передавати під керування іншим. Реалізація методу приведена нижче:

```
internal void StartToReadData()
{
      Network.BeginRead(BufferByte, 0, BufferByte.Length, new
AsyncCallback(EndRead), false);
      Handle.WaitOne();
}
```

Після того, як сокет отримує дані, починається виконання методу, який ці дані запише у буфер цього класу. Цей метод приватний і у класі має назву EndRead, в ході виконання програми, потік, який дійде до методу BeginRead не буде тормозити роботу програми, а очікуватиме на дані від сервера і після їх отримання продовжить виконання методу в якому був викликаний.

Аналогічним чином праюють методи для відправки повідомлень, різниця полягає лише у тому, що потрібно використати метод обертку, який запускатиме окремий Task, для відправки даних, цей метод має назву Send і його реалізація приведена нижче:

```
public void Send(byte[] packet)
{
    if (SendPacket == null || SendPacket.IsCompleted)
    {
        SendPacket = Task.Factory.StartNew(() => BeginWrite(packet));
    }
    else
    {
        SendPacket.ContinueWith(i => BeginWrite(packet));
    }
}
```

Метод BeginWrite являється недоступним поза цим класом і його функція полягає у виклику аналогічного методу BeginWrite з класу NetworkStream і завершеню передачі даних методом EndWrite. Повний код реалізації даного класу приведений у лістингу В.8.

Розпочинаємо реалізаціє клієнтського додатку для комп'юетра Спосіб реалізації патерну MVVM, для додатку цього типу був описаний у попередньому розділі. Тепер, розпочнемо розробку із реалізації сторінок авторизації і їх логіки.

Для початку виділемо основну форму, яка буде направляти нас на форму авторизації і не матиме зв'язку із нею. Код преведений нижче:

```
<Grid>
<Frame Name="mainFrame" NavigationUIVisibility="Hidden"/>
</Grid>
```

І виконуємо навігацію до сторінки авторизації наступним чином:

```
Navigate(new Uri("/pages/authorize/loginpage.xaml",
UriKind.RelativeOrAbsolute));
```

Розпочнемо розробку авторизації із компонента, який відповідає за розмітки сторінки. Реалізація приведена у лістингу В.15. Вигляд сторінок, представлений на рисунках Д.7 – Д.10. Головними елементами цих сторінок, є підключення моделі представлення, яка дозволить приєднувати змінні. Також, був написаний власний візуальний елемент і інтегрований у форму. Код підключення свого контролу наведений нижче:

```
<CustomCountrols:TextBoxWithPlaceHolder
Grid.Row="1"
PlaceHolder="Login"
DataContext="{Binding Path=LoginView.Login}"/>
```

В моделі представлення цієї сторінки, встановлюємо поля для авторизації у системі, також потрібні три команди, дві для переходу до інших сторінок і одна

для відправки пакету із даними на сервер і спроби авторизації. Лістинг коду із командою відправки даних на сервер наведений нижче:

```
private MainWindow mainWindow { get => Application.Current.MainWindow as
MainWindow; }
private void LoginCommandExecute(object obj)
{
mainWindow.mainFrame.Navigate(new Uri("/pages/mainpages/chatslistpage.xaml",
UriKind.RelativeOrAbsolute));
}
```

Аналогічним чином розроблені інші сторінки авторизації: реєстрація та відновлення паролю. Також є дві сторінки із якими користувач взаємодіє при підтвердежні реєстрації.

Розглянемо спосіб підключення до серверу, формування пакету, його відправка і отримка з подальшою обробкою даних на прикладі реєстрації нового облікового запису.

Для підключення до сервера на усіх сторінках використовується аналогічний методи, який приведений у лістингу коду нижче:

```
private EMTP.PC client;
client = new EMTP.PC();
client.PC = new TcpClient();
TCP.Connect(IPAddress.Parse(adress),port);
client.Network = obj.TcpClient.GetStream();
client.Data = new byte[65536];
```

Цей спосіб підключення має під собою створення розробленого модуля Client, який тримає у своїх полях дані про підключення і спроможний приймати і відправляти дані серверу. Розглянемо спосіб відправки даних на сервер, код придставлений нижче:

```
var loginPacket = Packet.Create(PacketType.LoginClient);
loginPacket.SetFields(new LoginClientFields() { Login = _loginView.Login.Text,
Password = _loginView.Password.Text });
var loginPacketBytes = loginPacket.ToPacket();
client.Send(loginPacketBytes);
```

У цьому фрагменті коду, чудово видно спосіб взаємодії із нашим протоколом передачі даних, формування пакету відбувається в декілька кроків: створюється пакет, куди передається тип пакету; типу пакета відповідає клас із його полями, який передається в якості параметру у метод SetFields; перетворення пакету у зашифрований масив байт. Спосіб отримання відповіді від серверу приведений у лістингу нижче:

```
if (client.TCP.Connected)
{
    try
    {
        client.BeginRead();
        if (client.Data.Length != 0)
        {
            var acceptPacket = Packet.Parse(client.Data);
            if (acceptPacket.Type == PacketType.AcceptServer)
            {
                var field = (AcceptServerFields)acceptPacket.GetFields();
                if (field.Answer == "DAROVA")
                {
                    var content = Content.GetContent();
                    content.client = obj;
                    client.Data = new byte[65536];
                    return true;
                }
            }
        }
    }
    catch
    {
        client.TCP.Client.Close();
        client.TCP.Close();
        return false;
    }
}
```

З коду можна зрозуміти, що при отримані неправильного типу пакету, який прописаний у сценарії цієї операції, відбувається відкючення від цього серверу, але якщо пакет був отриманий правильний, програма продовжує своє виконання, в даному випадку, перенаправляє користувача на головне вікно додатку.

Інші сторінки, які відповідають за авторизацію користувача, працюють за схожим сценарієм, лише відправляють інші пакети і для підтвердження, повинні отримати інший тип пакету.

Отже, ми розібрали спосіб побудови сторінок, які відповідають за авторизацію користувача, головним елементом, є передача у клас Content, об’єкта, який відповідає за підключення користувача до серверу. Тепер перейдемо до реалізації головного вікна додатку.

Дана сторінки, являється комплексним користувацьким інтерфейсом і включає в себе велику кількість елементів взаємодії та відображення контенту. Перерахуємо основні із них:

а) кнопки: налаштувань, пошуку користувачів, додавання користувачів, відправка повідомлення, вихід із акаунта;

б) списки: список чатів, список листування конкретних чатів, для кожно чату він свій;

в) текстові поля: поле для вводу повідомлення, поле для вводу ім’я користувача.

Взаємодіяти можна із усіма візуальними елементами. Цій сторінці, потрібне підключення моделі представлення, яке приведене нижче:

```
<Page.DataContext>
<vm:ChatsListViewModel/>
</Page.DataContext>
```

Для створення симпатичних іконок на сторінці, був використаний фреймворк Material Design. Приклад створення кнопки налаштувань, яка знахоидться на усіх сторінках приведений нижче:

```
<Button Grid.Row="1"
 FontWeight="SemiBold"
 Height="50"
 BorderThickness="0"
 Style="{StaticResource primarybutton}">
      <materialDesign:PackIcon Kind="Settings" Width="24" Height="24"/>
</Button>
```

При завантажені цієї сторінки, підключена модель представлення також ініціалізується і у конструкторі присвоюються необхідні поля для роботи

додатку, відбувається запуск потоку, який відповідає за отримання даних від серверу. Додаток працює у двох потоках. Конструктор моделі представлення приведений нижче:

```
public ChatsListViewModel()
{
    _chatsList = new ObservableCollection<Chat>();
    BindingOperations.EnableCollectionSynchronization(_chatsList, new object());

    var content = Content.GetContent();
    client = content.client;

    Start();
}
```

Тут завантажується список чатів, присвоюється об’єкт класу Client, для читання та відправки даних від серверу. Метод Start, запускає описаний раніше потік, зараз розглянемо цей метод, його код приведений нижче:

```
private void Start()
{
    Thread connectionFlow = new Thread(new ThreadStart(Connecting))
    {
        IsBackground = true
    };
    connectionFlow.Start();
}
```

Сам метод Connection, виконує роль слухача, який опрацьовує вхідні пакети від серверу і виконує функцію менеджера пакетів. У випадку виникнення виключення программи, клієнт відключається від серверу і повертається на сторінку авторизації. Код цього методу приведений нижче:

```
private void Connecting()
{
      try
      {
            while (client.TcpClient.Connected)
            {
                  client.BeginRead();
                  ClientTypeManager(client);
            }
      }
      catch
      {
```

```
            mainWindow.mainFrame.Navigate(new
Uri("/pages/фгерщкшяу/дщпштзфпу.xaml", UriKind.RelativeOrAbsolute));
            client.TcpClient.Client.Close();
            client.TcpClient.Close();
        }
}
```

Метод ClientTypeManager перетворює отримані дані у пакет протоколу, після чого перевіряє його тип і викликає метод обробки цього пакету.

Для відображення списків і прив’язки до них об’єктів, використовуються ObservableCollection, яка працює, як і метод OnPropertyChanged, повідомляючи представлення про зміну даних у ньому. Приклад створення такої колекції приведений нижче:

```
protected Collection<MyOwnMess> _mess;
internal Collection<MyOwnMess> Mess
{
    get
    {
        return _mess;
    }
    set
    {
        Set(ref _mess, value, nameof(Mess));
    }
}
```

Для того, щоб прив’язати цю колекцію до списку, потрібно в атрибутах цього елемента вказати SourceItem та SelectItem. Спосіб прив’язки цього списку показаний далі:

```
<ListBox
ItemsSource="{Binding Path=ChatsList, UpdateSourceTrigger=PropertyChanged}"
SelectedItem="{Binding Path=Chat, UpdateSourceTrigger=PropertyChanged}">
      <ListBox.ItemTemplate>
            <DataTemplate>
                  <StackPanel>
                              <TextBlock FontSize="14"
                                         Text="{Binding Path=UserName}"
                                         Margin="5 0"/>

                              <TextBlock Text="{Binding
Path=Messages[0].UserMessage}"
                                         Margin="5 0"
                                         TextTrimming="CharacterEllipsis"
                                         Opacity="0.6"
                                         FontSize="11"/>
                  </StackPanel>
```

```
            </DataTemplate>
        </ListBox.ItemTemplate>
</ListBox>
```

Щоб отримувати дані про вибраний об’єкт, потрібно аналогічним чином, у стилі MVVM, створити поле із класом, який відображає колекцію. Повний код моделі представлення головного вікна приведений у додатку В.16. Вигляд візуального інтерфейсу настільної програми приведений на рисунку Д.10. Говна сторінка показана на рисунку Д.11.

Отже, був розроблений настільний додаток, для операційної системи Windows. Була описана розробка сторінок для авторизації користувача і сторінка головного вікна із своїми особливостями. Тепер переходимо, до розробки мобільного додатку.

Оскільки додаток буде реалізований із використанням шаблону проектування MVVM, на початку потрібно створити базову структуру модулів і реалізувати стандартні інструменти, які дозволять повноціно використовувати даний підхід.

Детальні методи реалізації патерна були описані у попередньому розділі при детальному проектувані. Розпочнемо із створення представлення, як було спроектовано, ми використовуємо такий тип сторінки як Shell, тому для форм авторизації опишемо загальну сторінку із відповідними переходами і прив’язками моделей представлення, змінемо стартове вікно App.xaml.

У додатку В.9 буде представлений код розмітки нашої навігаційної сторінки авторизації. Головним фрагментом, являється підключення до папки із необхідними представленнями, що дозволить передавати потрібні вікна на представлені, лістинг коду наведений нижче:

```
xmlns:pages="clr-namespace:FastMessanger.Views.AuthorizeView"
```

Ми можемо звертатись до модулів наших форм, як вже було сказано, ми використовуємо одну сторінку, як головну і навігаційну по інших сторінках,

тобто TabMenu, спосіб додавання форми до навігаційного меню і її іменування наведений у лістингу нижче:

```
<ShellContent ContentTemplate="{DataTemplate pages:LoginPage}" Route="loginpage" Title="Login"/>
```

Таким способом, ми можемо звертатись і переходити до цієї сторінки використовуючи ієрархію. Shell, приклад переходу до цієї сторінки показаний у лістингу нижче:

```
Shell.Current.GoToAsync("//loginpage");
```

Також у нас є дві сторінки, перехід до яких потрібно виконувати лише після виконання відповідної умови і лише з призначених для цього форм, ми використаємо інший тег, саме FlyoutItem і вказавши атрибути IsVisible рівним FALSE, забороняємо користувачу переходити до цього вікна із будь-якої частини додатку, але вписуємо цю форму у ієрархію об’єкту Shell і аналогічним способом, який був показаний при переході до форми Login, можемо виконувати навігацію.

Отже, ми створили правильну навігацію і об’єднали форми у єдину ієрархію, тепер розпочнемо додавання необхідного функціоналу. Робити це будемо на прикладі форми авторизації та реєстрації, повний код форми приведений у лістингу В.10.

Головний елемент цієї форми, це підлючення її до відповідної моделі представлення. Підключення показано в лістингу кода нижче:

```
<ContentPage.BindingContext vm:LoginViewModel/>
```

Перед підключеннями ми створили сам клас для цієї моделі представлення і підключили тег із шляхом до цього об’єкту. Увесь наступний код розмітки представляє собою створення загальних стилів для однакових візуальних

елементів і Binding потрібних полів до еквівалентних їм в моделі представлення, спосіб приєднання показаний нижче:

```
<Entry Text="{Binding Login, Mode=TwoWay}"/>
```

Тут ми прив’язуємо зміну типу string, до елементу із яким взаємодіє користувач, шляхом вводу тексту. Атрибут Mode=TwoWay, говорить про те, що змінювати Text можна в розмітці і у коді, ці зміни будуть однакові для них. Аналогічним чином прив’язуються усі інші елементи.

Розпочнемо реалізацію моделі представлення і створення відповідного функціоналу. Отже, нам потрібно створити поля для прив’язаних елементів і потрібно, щоб вони відповідали контексту PropertyChanged, тобто при зміні даних буде повідомлятись представлення і його модель. Для цього, просто потрібно наслідувати нашу базову модель представлення. Реалізація приведена у лістингу В.1.

Після наслідування бозової моделі представлення і додавання простору імен, яке дозволить використовувати клас, який створює команду, переходимо до реалізації самої авторизації.

Отже, в методі команди нам потрібно викликати функцію, яка буде виконувати підключення до серверу і відправку даних йому. Метод буде повертати булеву зміну і якщо усе пройде успішно, ми будем перенаправленні на форму головного авторизованого вікна.

Для цього, ствоюється об’єкт класу, який був описаний у попередньому розділі і він потрібен для підключення до серверу. Код заповнення полів об’єкту приведений нижче:

```
mobile = new Mobile();
mobile.TCP = new TcpClient();
mobile.TCP.Connect(IPAddress.Parse("192.168.0.103"), 8888);
mobile.Network = mobile.TCP.GetStream();
mobile.Data = new byte[65536];
```

Тут у 3 рядку відбувається підключення до локального серверу і після цього можна отримати об’єкт NetwortkStream, якщо підлючення було вдалим, також ми заносимо цей код у конструкцію try…catch, щоб відловити можливі виключення і не викликати аварійне завершення додатку.

Після того, як ми приєдналися до серверу, відбувається створення пакету із типом LoginClient. У пакет передаються поля, які ввів користувач на представлені, його логін та пароль. Код створення пакету і його відправка і форматування описані далі:

```
var loginPacket = Packet.Create(PacketType.LoginClient);
loginPacket.SetFields(new LoginClientFields() { Login = _login, Password
= _password });
var loginPacketBytes = loginPacket.ToPacket();
mobile.Send(loginPacketBytes);
```

Після відправки даних на сервер, потрібно дочикатися його відповіді і обробити дані. Для цього, ми ще раз перевіряємо чи ми підключені до цього серверу і викликаємо метод BeginRead. Код представлений нижче:

```
mobile.StartRead();
```

Цей метод був описаний у попередньому розділі, з його допомогою ми не блокуємо основний потік, а очікуємо на повідомлення від серверу. Як тільки повідомлення надходить нам, ми перетворюємо його на пакету із типом AcceptServer. Виклик методу Parse, представлений нижче:

```
var acceptPacket = Packet.Parse(mobile.Data);
```

Перевіряємо поля на наявність ключа і якщо усе вірно, метод повертає значення TRUE і користувач перенаправляється на головну сторінку.

Аналогічним чином працюють інші сторінки і моделі представлення, які відповідають за авторизацію користувача. Для прикладу у лістингу В.11 буде

показаний код розмітки реєстрації, а у лістингу В.12 код моделі представлення цієї сторінки.

Отже, переходимо до кабінету користувача. Спосіб навігації буде використовуватись подібний до описаного в цьому розділі. Відмініть заключається у виборі візуальних елементів. У нас присутнє бокове меню, яке щоб побачити потрібно провести по екрану в право. Також, у випливаючому меню знаходиться кнопка для виходу з акаунта. Код розмітки представлений у лістингу В.13.

Навігаційне меню доступне у кожній формі і відповідно потрібно створити модель представлення і реалізувати кнопку виходу із меню, для цього ми використаємо патерн Singlton, в який передається об’єкт класу Client, що містить дані про підключення. Потрібно роз’єднати і направити навігацію додатку до попердніх форм. Код команди представлений нижче:

```
public Command LogoutCommand
{
    get
    {
        return new RelayCommand(LogoutCommandExecute,
CanLogoutCommandExecute);
    }
}

private void LogoutCommandExecute(object obj)
{
    var content = Content.GetContent();
    content.mobile.TCP.Client.Close();
    content.mobile.TCP.Close();
    Application.Current.MainPage = new AppShell();
}
```

Перейдемо до реалізації основного функціоналу. Коли користувач авторизувався, головна форма, яку він бачить і з якої продовжується робота додатку, це ChatViewsPage, оскільки старт форми починається у конструкторі моделі представленя цією форми при ініціалізації сторінки, то і запустити основний цикл, який буде очікувати повідомлення від серверу потрібно тут. Код конструктора моделі представлення знаходиться далі:

```
public ChatsViewsViewModel()
{
    var content = Content.GetContent();
    mobile = content. mobile;

    Start();
}
```

Метод Start запускає новий потік, який починає виконання методу Conneting, що відповідає за отримання повідомлень від серверу і не блокує форму користувача, його завдання обробляти вхідні пакети за допомогою методу ClientTypeManager. У випадку помилки підключення, з'єдання користувача із сервером переривається. Лістинг коду методу Connecting наведений нижче:

```
private void Connecting()
{
    try
    {
        while (mobile.TCP.Connected)
        {
            mobile.StartRead();
            ClientTypeManager(mobile);
        }
    }
    catch
    {
        mobile.TCP.Client.Close();
        mobile.TCP.Close();
    }
}
```

Після того, як подія EventWaitHandle просигналила, що дані отримані, викликається метод для розпізнавання пакету, після спроби його перетворити у пакет протоколу EMTP. Якщо все пройшло успішно, то виконуються прописані інструкції у методах, які обробляють пакети. Код методу менеджера наведений у додатку В.14. Отже, ми реалізували додаток для мобільних платформ, тепер можна переходити до його тестування.

3.3 Керівництво користувача

Після розробки програмної системи, потрібно: описати сценарії використання; приклад користування додатком; опис роботи функціоналу та умови при яких програмний продукт функціонуватиме правильно.

Ця програмна система використовує клієнт-серверну архітектуру і її основна функція, це передача повідомлень через мережу Інтернет, тобто, головною умовою для роботи додатку, являється підключення до Інтернету. Також, додаток для мобільних пристроїв не працюватиме у пасивному режимі, тобто ви не будете отримувати повідомлення про надходження нових листів від інших користувачів, а зможите їх прочитати лише авторизувавшись у ваш обліковий запис, аналогічно і для додатку на настільному пристрої.

Ваш пристрій повинен відповідати мінімальним вимогам для роботи додатку, що забезпечить комфортне користування. Опишимо основний сценарій роботи із додатком, а саме: реєстрацію і авторизацію.

Отже, поля для введення даних: логін, пароль, повідомлення, тощо, будуть виглядати ідентично на усіх сторінках додатку. Кнопки для виконання певної дії на формах авторизації будуть відображені в одному стилі. Вигляд форми авторизації можна подивитися на рисунку Д.1. У мобільних додатків знизу форми буде знаходитись TabMenu, за допомогою якого можна переходити до потрібних сторінок. Вигляд представлений на рисунку Д.1. Зараз розглянемо рискунок Д.2, у ньому знаходяться поля для введення даних і одне з полів де відбувається введення паролю, це поле не буде показувати, що ви ввели, це робиться для безпеки. Такий самий метод буде використовуватись повсюди в додатку де відбувається введення паролю. Перехід до сторінок, які представленні на рисунках Д.4 та Д.5, відбувається із відповідних сторінок, це допоміжні сторінки, які використовуються для остаточного затвердження даних реєстрації. Відкрити їх можна лише після коректного введення даних.

Принцип роботи, як для мобільних так і для комп’ютерних додатків, стосовно авторизації, однаковий і інтуїтивний.

Розглянемо способи роботи із програмою після авторизації. Принцип роботи для обох платформ однаковий, відрізняється лише розміщення візуальних елементі, але користувачу на мобільному додатку буде зрозуміло, як працювати із настільним додатком і навпаки.

Після авторизації, користувачу відразу буде відображено список активних чатів, зліва буде знаходитись панель керування, справа поле для пошуку користувачів. Натиснувши на один активний чат із списку, користувач побачить усе листування і зможе обмінюватись листами.

3.4 Вимоги до технічних та програмних засобів

У вимогах до програмної системи зазначено, що додатки будуть працювати на мобільних пристроях із операційною системою Android і IOS. Тому, смартфони повині відповідати мінімальному технічному забезпеченю, яке дозволить використовувати ці операційні системи і тим самим додаток працюватиме коректно.

Опишемо мінімальні та рекомендовані системні вимоги для мобільних пристроїв із операційною системою Android у таблицях 3.1, 3.2.

Таблиця 3.1 – Мінімальні системні вимоги для Android пристроїв

Операційна система	Android 5
Кількість оперативної пам’яті	512 МБ
Вбудована пам’ять	8 ГБ
Частота процесора	1.1 GHz
Архітектура	32-bit

Таблиця 3.2 – Рекомендовані системні вимоги для Android пристроїв

Операційна система	Android 9
Кількість оперативної пам’яті	2 ГБ
Вбудована пам’ять	16 ГБ
Частота процесора	1.4 GHz
Архітектура	64-bit

Отже, були описані технічні вимоги для мобільних додатків із операційною системою Android. Щодо телефонів на базі IOS, конкретних вимог до технічного засобу немає, оскільки телефони мають стандартизоване технічне забезпечення і працюють коректно при довільній збірці комплектуючих затверджених компанією Apple, тому додаток працюватиме коректно на усіх пристроях із операційною системою IOS.

Опишемо технічні вимоги для комп’ютерів. Оскільки при розробці додатку використовувалась технологія WPF, яку можна використовувати на операційних системах Windows Vista і до самої останьої версії. Встановлення і використання не підтримується на операційній системах Linux. Також WPF не буде працювати на операційній системі macOS. В таблицях 3.3, 3.4 описані мінімальні та рекомендовані технічні вимоги до апаратної частини.

Таблиця 3.3 – Мінімальні системні вимоги для настільних пристроїв

Операційна система	Windows Vista
Кількість оперативної пам’яті	1 ГБ
Процесор	Частота більше 1 GHz
Виділена пам’ять	16 ГБ
Архітектура	32-bit

Таблиця 3.4 – Рекомендовані системні вимоги для настільних пристроїв

Операційна система	Windows 10
Кількість оперативної пам’яті	2 ГБ
Процесор	Більше або рівний 1 GHz, SoC
Виділена пам’ять	20 ГБ
Архітектура	64-bit

Отже, були описані технічні вимоги апаратної частини для мінімальної та комфортної роботи програмної система. Встановлення доддаткових програмних засобів відбуватиметься при інсталюванні програмного продукту, тобто користувачу не потрібно встановлювати додаткове ПО, інсталятор завантажить потрібні файли і бібліотека самостійно.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вибір та обґрунтування методів тестування

Проведення перевірки програми, важлива частина розробки. Тестування обов'язкова частина розробки програмної системи, метою тестування є:

– характеристики програмної системи, яка покаже відповідність між отриманою системою та заявленою;

– дослідження системи для визначення якості додатку;

– спостерігання за програмною системою, перевірка заявлених вимог і отриманих, перевірка реалізації заявлених функцій системи.

Тестування, також, включає в себе аналіз і планування, розробку тестових сценаріїв, отримання звіту, проведення аналізу системи і виконання самих тестів. Ціль проведення тестування заключається в підвищенні якості системи, виявленні дефектів.

В основному, при тестуванні ми перевіряємо блоки коду на працездатність і контролюємо процес роботи програми, вивчаємо отримані дані на відповідність нашим очікуванням і робимо висновки. Існують різні підходи до тестування, але усі вони направлені на досягнення однієї мети: виявлення несправностей, усунення помилок програми, перевірка продуктивності системи.

Опишемо основні тестування програмного продукту:

– тестування окремих компонентів додатку;

– тестування компонентів системи;

– тестування при отримані.

Тепер розглянемо більш детально методи тестування. Програмна система перевіряється цими тестами в описаному порядку.

Модульне тестування, аналізує роботу програми на об'єктному рівні. Тестуються частини програмної системи: логічні файли, бібліотеки, програмний інтерфейс. Оскільки, шаблоном проектування був вибраний MVVM, то модульні тести чудово підходять для таких додатків. Тестувати програми розроблені таким

шаблоном дуже комфортно, що надає велику звітність, уся програма розбита на модулі. Модульне тестування вимагає від тестувальника знання програми і усіх модулів системи, тому тестування проводиться самим розробником додатку. Розробник створює код-тести для перевірки і відпрацювання шаблонів використання додатку.

При інтиграційному тестуванні проводиться перевірка окремих модулів, які перевірялись модульними тестами, шляхом інтеграції їх і перевірки помилок. Цей метод тестування спрямований на виявлення проблем із користувацьким інтерфейсом. Виконання тестування відбувається через проходження архітектурної структури системи, тобто "зверху-вниз" або із малих частин коду.

Системне тестування перевіряє усю програмну систему на помилки і несправності, проводиться кооперування апаратної частини із програмною.

Приймальне тестування, останій етап перевірки програмної системи, який дає відповідь на питання: чи відповідає програмна система заявленим вимогам? Тестування проводиться розробником та потенційними користувачами системи.

Під час опису основних методів тестування, було сказано про тестування «чорної скриньки», це вид тестування, який формує принцип тестування програмного продукту.

Спосіб перевірки «чорної скриньки», цей вид тестування передбачає, що ми не знаємо, як працює системна логіка. Проводиться тестування користувацької взаємодії із системою, вводяться дані і отримуються вихідні дані, це також називають функціональним тестуванням.

Спосіб перевірки «білої скриньки», цей вид враховує способи і логіку побудови програмної системи. Щоб проводити таке тестування, потрібно знати і розуміти внутрішню реалізацію додатку, тобто, таке тестування проводить сам розробник системи. Можна проводити перевірку окремих частин коду.

Спосіб перевірки «сірої скриньки», цей вид при якому тествальник частково володіє інформацію про реалізацію додатку, але цього достатньо для проведення перевірки програми, можна сказати, що це середина між тестуванням чорного і сірого ящика, де ведеться документація.

Для нашого додатку і вибраних технологій розробки, чудово підходять юніт-тести, ними можна протестувати окремі блоки коду незалежно від інших компонентів системи, що чудово вписується в шаблон проектування MVVM. Можна робити перевірку сценаріїв очікування та результату і це чудово компонується із видом тестування «білої скриньки».

Для юніт-тестів характерними ознаками є:

– перевірка невеликих блоків коду, ми створюємо юніт-тести для перевірки невеликих частин коду, це може бути метод або клас. Тестування малих блоків дає змогу в швидкому написані великої кількості тестів.

– тестування окремих компонентів системи, моделей, представлень, ізольованих частин додатку, надає можливість виявити помилку в окремих компонентах.

– автоматичне тестування, тобто результатом тестування повинні бути прості розв’язки і для цього використовуються фреймворки.

– тестуються лише загально досяжні комноненти додатку, щоб запобігти виникненю провалу тестів через зміну реалізації модулів.

Беручи до уваги вибрану мову програмування і середовища для розробки, а саме Visual Studio, доцільно використовувати такі фреймворки:

– MS Test: фреймворк для юніт-тестів розроблений компанією Microsoft, який має підтримку .NET Core;

– NUnit: фреймворк заснований на Junit, але портований на .NET;

Існує також метод перевірки програмного продукту, точніше методологія для розробки TDD – розробке через тестування, але ми її використовувати не будемо, оскільки тестування проводиться на готовій програмній системі.

Також варто сказати про нефункціональні тести, які доцільно провести для аналізу роботи програмної системи: тестування надійності додатку, тестування на сумісність із платформами та тестування ефективності коду.

Тестування надійності у нашій системі важлива частина, оскільки частина роботи покладається на захист програмної системи і забезпечення надійності передачі і отримання даних.

Тестування на сумісність потрібно провести тому, що розроблюваний програмний продукт повинен працювати на двох платформах: комп'ютерна та мобільна, а також працювати на різних видах операційних систем.

Отже, була описана мета перевірки програмного продукту, визначена ціль тестування, проаналізовані методи тестування додатку, описані види тестування систем та вибрані технології для проведення тестів і описані доступні фреймворки для проведення тестування. Результатом аналізу боли визначено, що будуть проведені такі методи тестування: модульне, інтеграційне, тестування систем управління та приймання. Для нашого проекту чудово підходить вид тестування «білого-ящика», але доцільно провести також перевірку за допомогою «чорного-ящика», щоб розуміти результат роботи користувача із програмою. Фреймворк вибраний MS Test.

4.2 Доведення працездатності програмного продукту

Розпочнемо тестування компонентів програмного продукту із модульного тестування і перевірки: правильності роботи додатків, коректність обробки вхідних та вихідних даних. Сформуємо таблиці із сценаріями тестування, які будуть мати стовпці: назва тестового сценарія; модуль, який тестується; метод модюля, який тестується; дані на вхід; очікуваний результат, по завершенні роботи конкретного модуля. Після чого проведемо функціональне тестування системи, аналогічно створимо таблиці функціональних вимог додатків. В кінці проведемо системне тестування працездатності мобільного та настільного додатку на відповідність заявленим технічним вимогам.

У таблиці 4.1 приведені основні тестові сценарії роботи протоколу, це модулі та методи, через які відбувається взаємодія із протоколом.

Після проведення основного тестування модулів, потрібно провести тестування внутрішніх, закритих, компонентів протоколу, які відповідають за обробку даних. У таблиці 4.2 приведене сценарії модульного тестування.

Таблиця 4.1 – Тестові сценарії відкритих модулів протоколу

Назва тестового сценарія	Модуль	Метод	Дані на вхід	Очікуваний результат
Перетворення у пакет	Packet	ToPacket	Об’єкт класу Packet із заповненими полями Type і MainField	Зашифрований масив байтів
Перетворення у пакет	Packet	Parse	Зашифрований масив байтів	Об'єкт класу Packet із полями Type і MainField
Встановити поля пакету	Packet	SetFields	Дочірній клас успадковиний від MainField	Поле MainField створеного пакету
Отримати поля пакету	Packet	GetFields	Вхідних даних немає	Клас MainField
Створити пакет протоколу	Packet	Create	Передати у метод тип пакету	Пакет із вказаним типом
Перетворити масив байт у строку	Packet	ByteArray ToString	Масив байт	Перетворена строка
Перетворити строку у масив байт	Packet	StringToB yteArray	Строка	Масив байт
Перетворити масив байт у структуру	Packet	ByteArray ToFixedO bject	Масив байт	Структура даних
Перетворити структуру у масив байт	Packet	FixedObje ctToByte Array	Структура даних	Масив байт

Таблиця 4.2 – Тестові сценарії закритих модулів протоколу

Назва тестового сценарія	Модуль	Метод	Дані на вхід	Очікуваний результат
Парсинг полів пакету	MainField	ParseFields	Масив байт пакету, масив байт ключа	Масив байт, який представляє собою поле пакету
Перевірка заголовків полів пакету	MainField	CheckKeys	Масив байт, масив ключа	Булеве значення
Перетворення в неопізнаний пакет	Packet	ParseUnknown	Немає	Повертає пакет із типом Unknown

Також кожен тип пакету має свій клас, у якому знаходяться його поля, ці класи успадковані від класу MainField і мають його відкриті методи: Parse, ToPacketField. Класи, які успадкували батьківський, реалізують власні алгоритми

по перетворенні і розшифрувані полів. Приклад тестування одного із такого класу приведений в таблиці 4.3.

Таблиця 4.3 – Тестові сценарії перевірки класу із даними пакету

Назва тестового сценарія	Модуль	Метод	Дані на вхід	Очікуваний результат
Парсинг полів пакету	AcceptPacket	Parse	Масив байт, клас пакету	Пакету із заповненими полями
Перевірка заголовків полів пакету	AcceptPacket	ToPacketField	Сформований масив із встановленими заголовками пакету	Масив байт, який доповнився полями пакету і його заголовками

Оскільки протокол являється бібліотекою, а не повноцінним додатком, проводити функціональні тести не потрібно. Отже, були сформовані сценарії тестування протоколу передачі даних, де протестовані модулі протоколу.

Розпочнемо тестування із авторизації та реєстрації користувача. Нижче у таблиці 4.4 приведена сценарії роботи авторизації настільного додатку.

Таблиця 4.4 – Тестові сценарії авторизації настільного додатку

Ідентифі-катор	Компонент	Модуль	Дані на вхід	Очікуваний результат
1	2	3	4	5
PC.L1	LoginPage	Авторизація	Ввести дані в текстові поля, натиснути на кнопку	Валідація вхідних даних, формування пакету, відправка даних на сервер, перехід до основної сторінки
PC.L2	LoginPage	Кнопка реєстрації	Натистути на кнопку	Перехід до сторінки реєстрації
PC.L3	LoginPage	Кнопка відновлення паролю	Натиснути на кнопку	Перехід до сторінки відновлення

Кінець таблиці 4.4

1	2	3	4	5
PC.R1	RegisterPage	Реєстрація	Заповнити поля реєстрації, натиснути на кнопку реєстрації	Валідація вхідних даних, формування пакету, відправка даних на сервер, перехід до сторінки авторизації
PC.R2	RegisterPage	Кнопка авторизації	Натиснути на кнопку авторизації	Перехід до сторінки атворизації
PC.R3	RegisterPage	Кнопка відновлення паролю	Натиснути на кнопку відновлення	Перехід до сторінки відновлення
PC.F1	ForgotPassPage	Відновлення паролю	Заповнити поля, натиснути на кнопку	Формування пакету, відправка даних на сервер, перехід до сторінки заміни пароля
PC.F2	ForgotPassPage	Кнопка авторизації	Натиснути на кнопку авторизації	Перехід до сторінки атворизації
PC.F3	ForgotPassPage	Кнопка реєстрації	Натистути на кнопку	Перехід до сторінки реєстрації

Отже, були описані сценарії модульного тестування для сторінок авторизації користувача. Тепер розпочнемо тестування основної сторінки. Нижче у таблиці 4.5 приведені сценарії модульного тестування головної сторінки додатку для настільної системи.

Таблиця 4.5 – Тестові сценарії головної сторінки настільного додатку

Ідентифі-катор	Компонент	Модуль	Дані на вхід	Очікуваний результат
1	2	3	4	5
PC.M1	ChatListPage	Вийти із акаунту	Натиснути на кнопку	Перенаправити на сторінку авторизації і від'єднати від серверу
PC.M2	ChatListPage	Налаштування	Натистути на кнопку	Перехід до сторінки із інформацією про обліковий запис
PC.M3	ChatListPage	Пошук користувачів	Ввести дані і натиснути кнопку	Відобразити результат пошуку
PC.M4	ChatListPage	Добавити чат	Натиснути кнопку	Добавити чат до загального листа чатів.

Кінець таблиці 4.5

1	2	3	4	5
PC.M5	ChatListPage	Перейти до певного чату	Натиснути на ім'я користувача із списку чатів	Відобразити переписку із цим користувачем
PC.M6	ChatListPage	Відправити повідомлення	Написати повідомлення у текстовий елемент і натиснути кнопку	Надіслати дані на сервер, відобразити написане повідомлення у формі листування
PC.M7	ChatListPage	Видалити чат	Натиснути на чат після чого натиснути на кнопку хрестик	Запитати користувача чи дійсно він бажає видалити чат, видалити чат

Тепер проведемо функціональне тестування додатку і розпочнемо із перевірки необхідних елементів, які відповідають за обробку авторизації, будуть описані сценарії роботи. Після чого проведемо перевірку основного функціоналу. Нижче у таблиці 4.6 приведене функціональне тестування.

Таблиця 4.6 – Функціональне тестування настільного додатку

Назва тестового сценарія	Дія	Дані на вхід	Очікуваний результат
PC.A1	Авторизація	Користувач вводить дані у поля і натискає кнопку авторизації	Якщо дані введені правильно, користувач направляється на головне вікно
PC.A2	Реєстрація	Користувач вводить дані у поля і натискає кнопку	Якщо дані введені правильно, користувач направляється на вікно авторизації
PC.A3	Відновлення паролю	Користувач вводить дані у поля і натискає кнопку	Якщо дані введені правильно, користувач направляється у вікно введення нового паролю

Функціональне тестування проводилось на трьох комп’ютерах із різними технічними властивостями, але усі вони були вищими за рекомендовані. Додаток показав хорошу продуктивність роботи на усіх пристроях.

Аналогічним чином, розпочнемо тестування із авторизації та реєстрації користувача. Нижче у таблиці 4.7 приведена тестові сценарії роботи мобільного додатку. Тестування переходу між формами авторизації, аналогічне, як і при тестувані настільного додатку, описувати не потрібно.

Таблиця 4.7 – Тестові сценарії мобільного додатку

Ідентифі-катор	Компонент	Модуль	Дані на вхід	Очікуваний результат
M.L1	LoginView	Авторизація	Ввести дані у поля, натиснути на кнопку	Валідація даних, формування пакету, відправка даних, перехід до основної форми
M.R1	RegisterView	Реєстрація	Ввести дані у поля, натиснути на кнопку	Валідація даних, формування пакету, відправка даних на сервер, перехід до форми авторизації
M.F1	ForgotPassView	Відновлення паролю	Ввести дані, натиснути на кнопку	Валідація даних, формування пакету, відправка даних, перехід до форми авторизації

Отже, були описані сценарії модульного тестування для форм авторизації користувача. Тепер розпочнемо тестування основної форми. Нижче у таблиці 4.8 приведені сценарії модульного тестування головної форми додатку.

Таблиця 4.8 – Тестові сценарії основного функціоналу мобільного додатку

Ідентифі-катор	Компонент	Модуль	Дані на вхід	Очікуваний результат
1	2	3	4	5
M.M1	ChatsView	Перейти до чату	Натиснути у списку чатів на потрібний	Перенаправити користувача на форму листування
M.M2	ChatsView	Новий чат	Додаток отримує дані від користувача	Створення чату, відображення його на головній формі
M.M3	SearchUserView	Пошук користувача	Ввести дані у текстове поле та натиснути кнопку	Надіслати дані, отримати відповідь, відобразити знайденого користувача

Кінець таблиці 4.8

1	2	3	4	5
М.М4	SearchUserView	Додати новий чат	Натиснути на ім'я знайденого користувача	Перенаправити на головну форму із поновленням списку чатів
М.М5	DetailChat	Відправити повідомлення	Написати повідомлення у полі і натиснути на кнопку	Відправити дані із повідомленням на сервер, відобразити це повідомлення у формі листування
М.М6	DetailChat	Отримати повідомлення	Немає	Отримати повідомлення від користувача і відобразити його у формі листування
М.М7	AccountView	Змінити пароль	Натиснути на кнопку	Перенаправити на форму зміни паролю
М.М8	AccountView	Змінити ім'я	Натиснути на кнопку	Перенаправити на форму зміни ім'я
М.М9	MainShell	Вийти із акаунта	Натиснути на кнопку	Роз'єднати клієнта із сервером і перенаправити на форму авторизації

Тепер проведемо функціональне тестування додатку. Далі у таблиці 4.9 приведене функціональне тестування настільного додатку.

Таблиця 4.9 – Функціональне тестування настільного додатку

Назва тестового сценарія	Компонент	Дані на вихід	Очікуваний результат
М.А1	Авторизація	Користувач вводить дані у поля і натискає кнопку авторизації	Відбувається перехід до основної форми
М.А2	Реєстрація	Користувач вводить дані у поля і натискає кнопку	Відбувається перехід до форми авторизації
М.А3	Відновлення паролю	Користувач вводить дані у поля і натискає кнопку	Відбувається перехід до форми введення паролю

Щоб краще зрозуміти, як працює головне вікно додатку, потрібно провести його функціональне тестування, яке наведене у таблиці 4.10. Тут буде проведено основні базові сценарії, які описані в інструкції користувача, ці сценарії

покриють основний функціонал, який зустріне користувача під час користування додаток. Хоча схожість із ком'юетрною версією існую, але через специфіку реалізації цього додатку, сценарії будуть відрізнятись.

Головне вікно складається із списку чатів, вспливаючого меню та кнопок керування, які знаходяться на кожному із вікон.

Таблиця 4.10 – Функціональне тестування головного вікна

Назва тестового сценарія	Дія	Дані на вхід	Очікуваний результат
М.А4	Перейти до іншого вікна	Користувач користуючись нижнім меню баром, вибирає форму до якої хоче перейти	Користувач переходить до вибраної форми
М.А5	Знайти користувача	Користувач вибирає у нижньому меню барі форму пошуку, користувач вводить дані у поле для пошуку і натискає кнопку	Користувач переходить до форми пошуку. У списку знайдених користувачів відображається результат пошуку.
М.А6	Додати новий чат	Клієнт натискає на ім'я знайденого користувача	Користувач переходить до головного вікна де відображається новий чат
М.А7	Відправка повідомлення	Користувач натискає на доступний чат, користувач вводить повідомлення у поле, користувач натискає кнопку відправити.	Користувач переходить до листування. У формі відображається повідомлення, яке користувач відправив.
М.А8	Отримати повідомлення	Користувач очікує повідомлення	У формі листування відображається отирмане повідомлення
М.А9	Видалити чат	Користувач вибирає чат, користувач натискає на кнопку видалити чат.	Чат видаляється і користувач переходить до оновленої форми головного вікна.
М.А10	Налаштування	Користувач виконує "свайп" вправо і вибирає пункт меню	Відображається бокове меню, користувач направляється на вибрану форму
М.А11	Вийти із акаунта	Користувач виконує "свайп" вправо і вибирає кнопку із написом “Вийти”	Відображається бокове меню, виконується вихід із облікового запису і перехід до форми авторизації

Для функціонального тестування, використані емулятори, що імітують реальні мобільні пристрої та власний телефон із операційною системою Android.

Розпочнемо тестування серверу, буде проведена перевірка правильності обробки вхідних даних від клієнтів і відправка їм повідомлень. Нижче у таблиці 4.11 приведена тестові сценарії роботи серверу.

Таблиця 4.11 – Тестові сценарії серверу

Ідентифі-катор	Компонент	Модуль	Дані на вхід	Очікуваний результат
S.M1	ServerSetting	Запустити сервер	Ввести адрес і порт у текстові елементи	Запустити сервер на отриманому адресі
S.M2	Log	Результати роботи	Немає	Виведення результату роботи із клієнтами
S.M3	ServerLogic	Listen	Об’єкт ServerFields	Запуск потоку
S.M10	ServerLogic	CheckConnection	Об’єкт Client	Перевірити чи клієнт онлайн
S.M11	ServerLogic	ServerTypeManager	Об’єкт Client	Розпізнати тип вхідного пакету і викликати відповідний модуль
S.M12	ServerLogic	Handle[Packet.Type]	Об’єкт пакету, об’єкт Client	Виконати обробку пакету і відправляє відповідь клієнту

Тепер перевіримо допоміжні модулі серверу, у табилці 4.12 наведені тестові сценарії цієї перевірки.

Таблиця 4.12 – Тестові сценарії додаткових модулів серверу

Ідентифі-катор	Компонент	Модуль	Дані на вхід	Очікуваний результат
1	2	3	4	5
S.M4	ServerLogic	AcceptClient	Об’єкт TCP, клас Client	Заповнити поля класу Client
S.M5	ServerLogic	LoginClient	Об’єкт Client	Менеджер вхідного пакету, який визначить наступний модуль обробки підключення

Кінець таблиці 4.12

1	2	3	4	5
S.M6	ServerLogic	AutoriseClient	Об’єкт пакету, об’єкт Client	Провести авторизацію клієнта
S.M7	ServerLogic	ForgotPassdClient	Об’єкт пакету, об’єкт Client	Провести зміну пароля клієнта
S.M8	ServerLogic	Connect	Об’єкт Client	Почати прослуховування клієнта

Функціональне тестування проводилось на комп'ютерах, які використовувались для тестування додатку. Звіт показаний у таблиці 4.13.

Таблиця 4.13 – Функціональне тестування серверного додатку

Назва тестового сценарія	Компонент	Дані на вихід	Очікуваний результат
S.A1	Запустити сервер	Адміністратор вводить дані про адрес серверу та його порт і натискає на кнопку запуску	Сервер запускається, користувач бачить відповідний напис, що сервер працює, відображається навантаження на процесор
S.A2	Зупинити сервер	Адміністратор натискає на кнопку зупинити сервер	Сервер вимикається і це супроводжується виведенням тексту про зупинку серверу.
S.A3	Моніторинг роботи серверу	Користувач натискає на одну із вкладок, які показують інформацію про роботу серверу	Користувач потрабляє на вкладку, якщо робота сервера викликала певну дію, це відображається у вигляді тексту у вкладці

Отже, було проведено тестування та перевірка програмної системи, в ході якого, були протестовані основні модулі додатків, перевірені функціональні сценарії, які відповідають технічним вимогам програмного продукту. Тепер переходимо до аналізу отриманих результатів.

4.3 Аналіз результатів тестування

Оскільки, розроблений протокол передачі даних, являється бібліотекою, то для нього були проведені лише модульні тести, щоб перевірити правильність роботи. За результатами тестування, не було виявлено проблем у роботі протоколу, усі модулі працюють правильно. Додавання нового типу пакету проходить етап створення нового класу із ім’ям з приставкою Field, що вносить новий тип у менеджер пакетів.

Методи, які відповідають за перетворення об’єкту у зашифрований масив байтів працюють правильно. Методи для дешифрування масиву виконують правильне розшифрування масиву і у випадку отримання неправильного пакету, відбувається його визначення, як невідомого.

Отже, за результатами тестування, протокол працює правильно, підключені клієнти обробляються коректно. Провівши модульне тестування додатків, виявлено спільний баг, який актуальний на обох платформах, він пов’язаний із елементом користувацького інтерфейсу, а саме, списком чатів. Після проведення аналізу, виявлені причини помилки і виконано виправлення несправності. Інші модулі додатків працюють справно і виконують поставленні задвання. Правильність виконання модулів приведене у таблиці 4.14.

Таблиця 4.14 – Звіт тестування додатків

Назва модуля	Платформа	
	Комп’ютер	Телефон
Авторизація	True	True
Реєстрація	True	True
Відновлення даних	True	True
Відправка повідомлення	True	True
Отримання повідомлення	True	True
Пошук користувачів	True	True
Додавання користувачів	True	True

Оскільки, модульне тестування показало правильність роботи додатків, то і результати функціонального тестування коректні. Додатки відповідають поставленим функціональним вимогам.

Щодо рекомендованих технічних вимог, додатки показали чудовий результат у роботі із вищими апаратними засобами ніж рекомендовані.

Також, було проведено тестування на мінімальних технічних вимогах, шляхом створення відповідного мобільного емулятора. Додаток показав хорошу швидкодію при кількості оперативної пам’яті 512 МБ і процесорі із частотою 1.0 Ghz. Оскільки телефони на операційній системі IOS, завжди відповідають рекомендованим вимогам, проводити тестування не потрібно.

Комп’ютери, для тесту, були вибрані із різними конфігураціями, але усі вони перевищували рекомендовані вимоги. Зараз 99% комп’ютерів і не приблизяться, до мінімальних техніних вимог, тому такий вибір обгрунтований. Додаток показав чудову швидкодію і чудово відобразив створений користувацький інтерфейс.

Отже, додатки працюють справно на усіх потрібних платформах, виконують поставленні завдання і відповідають технічним вимогам.

Після модульного тестування, було встановлено, що сервер генерує виключення при роботі. Якщо підключається користувач із операційною системою Android або IOS, і виконує вихід із додатку, головний потік серверу перенавантажується і на слабшій системі використовує 99% ресурсів центрального процесора, що робить неможливим подальшу обробку клієнтів і роботу серверу.

Причина такої поведінці серверу, полягає у тому, що мобільний додаток може працювати у пасивному режимі і фактично ніколи не вимикається, що і перенавантажує потік, хоча повинно було відключати клієнта. Для вирішення цієї проблеми, було створено сценарій, при якому сервер перевіряє чи користувач зараз активний, якщо сервер не отримує відповіді він роз’єднує клієнта.

Після усунення багу, сервер працює правильно і виконує поставлені завдання. У таблиці 4.13 приведені результати модульного тестування.

Табилця 4.15 – Результати модульного тестування серверу

Модуль	Результат проходження тесту
Запустити сервер	+
Listen	+
AcceptClient	+
LoginClient	+
AutoriseClient	+
RegisterClient	+
ForgotPassdClient	+
Connect	+
CheckConnection	+
ServerTypeManager	+
Handle[Packet.Type]	+

Отже, за результатами тестування, було виправлено помилку роботи серверу, виправлено несправності. Встановлено, що модулі сервера стали працювати правльно і що сервер виконує поставлені завдання.

За результатами перевірки роботи програмного продукту, було перевірено: настільний та мобільний додаток на відповідність функціональним вимогам, перевірена робота серверу, проведене модульне тестування протоколу передачі даних. Усі складові програмної системи працюють справно, виконують відведені їм завдання. Вимоги до технічних засобів, були визначені правильно і тестування на різних пристроях це показало. На рисунку Е.1 показаний результат тестування протоколу. На рисунках Е.2 та Е.3 результати тестування клієнтських та серверного додатку. За результатами програмна система працює правильно.

ВИСНОВКИ

Спочатку було проведено дослідження предметної області, в результаті якого з’явилося бачення специфіки роботи даного направлення. Після чого, проведений аналіз наявного програмного забезпечення, в результаті стали зрозумілі недоліки і переваги систем. Підсумком дослідження, стало виведення потреб до програмного продукту, які потрібно виконати. Щоб краще розуміти поставлену мету, розроблено технічне завдання.

Наступним кроком стало проектування програмного забезпечення. Проектування почалося з аналізу архітектури для розробки додатків та визначення загального шаблону системи. В результаті, було вибрано двухрівнену клієнт-серверну архітектуру із товстим клієнтом, шаблон проектування MVVM. Після окреслення підходів до розробки, було проведено детальну декомпозицію системи, де проведено проектування бази даних та опис інтерфейсу взаємодія для користувача. На основі проведеного проектування та декомпозиції вибрано засоби реалізації системи, якими стали: WPF, для настільного додатку; Xamarin, для мобільного додатку; власний протокол передачі даних EMTP.

Після визначення вимог і засобів реалізації, наступив етап розробки програмної системи. На початку, проведене детальне проектування модулів, в якому описані базові процеси реалізації шаблонів та архітектури і спроектовані основні компоненти додатку.

Програмна реалізація, де описана і представлена логіка роботи додатків, способи побудови клієнтської частини, процес взаємодії візульного інтерфейсу із модулями, алгоритм обміну даними сервера із клієнтом.

На основі розробленої програмної системи, описане керівництво для користувача та сформульовані вимоги до програмних та технічних засобів.

Завершальним етапом розробки програмного продукту, стало тестування і аналіз отриманих результатів. На початку, було вибрано засоби тестування, ними стали модульні тести та функціональне тестування. Після вибору методів тестування, перевірено програмну систему на відповідність поставленим

вимогам, де було виявлено незначну проблему в роботі сервера, несправність було усунено. Аналіз результатів показав, що програмне забезпечення працює правильно, що доводить працездатність системи.

Отже, за результатами диплоного проектування була розроблена програмна система обміну повідомленнями з використаням стійких алгоритмів шифрування, яка задовільняє поставлені вимоги і працює справно.

ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАННЯ

1. Л. П. Бедратюк. Дипломний проект : методичні вказівки щодо його виконання для студентів спеціальності 121 «Інженерія програмного забезпечення» / Л. П. Бедратюк, Г. І. Радельчук, Ю. В. Форкун, О. М. Яшина. – Хмельницький: ХНУ, 2020. – 77 с.

2. Обзор мессенджеров. Лучшие и популярные Интернет-мессенджеры [Електронний ресурс] / VoipOffice. – Режим доступу до ресурсу: https://www.voipoffice.ru/tags/messendzhery/.

3. Анисимов В. В. Диаграмы вариантов использования [Електронний ресурс] / В. В. Анисимов. // Sites.Google. – Режим доступу до ресурсу: https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema12/tema12_2

4. Клиент-серверная архитектура в картинках [Електронний ресурс] // Habr. – Режим доступу до ресурсу: https://habr.com/ru/post/495698/.

5. Паттерн MVVM [Електронний ресурс] / Metanit. – Режим доступу до ресурсу: https://metanit.com/sharp/wpf/22.1.php.

6. Олифер В. Компьютерные сети. Принципы, технологии, протоколы / В. Олифер, Н. Олифер. – Санкт-Петербург: Питер, 2016. – 992 с.

7. Мартин Р. С. Принципы, паттерны и методики гибкой разработки на языке C# / Р. С. Мартин, М. Мартин. – Санкт-Петербург: Символ-Плюс, 2011. – 768 с.

8. Фейт С. Архитектура, протоколы, реализация (включая IP версии 6 и IP Security) / Сидни Фейт. – Москва: Лори, 2000. – 450 с.

9. David B. M. TCP/IP Sockets in C#: Practical Guide for Programmers / B. M. David, J. D. Michael, L. C. Kenneth. – San Francisco: Elsevier, 2004. – 188 с.

10. Рихтер Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# / Дж Рихтер. – Санкт-Петербург: Питер, 2012. – 928 с.

11. Скит Д. C# для профессионалов: тонкости программирования / Джон Скит. – Москва: Вильямс, 2014. – 608 с.

12. Мартин Р. чистый код: создание, анализ и рефакторингю / Роберт Мартин. – Санкт-Петербург: Питер, 2019. – 464 с.

13. Сетевое програмированние для професионалов / [Э. Кровчик, В. Кумар, Н. Лагари та ін.]. – Москва: Лори, 2005. – 417 с.

14. Прайс Марк Д. C# 7 и .NET Core. Кросс-платформенная разработка для профессионалов / Дж Прайс Марк. – Санкт-Петербург: Питер, 2018. – 640 с.

15. Руководство по WPF [Електронний ресурс] / Metanit. – Режим доступу до ресурсу: https://metanit.com/sharp/wpf/.

16. Руководство по программированию для Xamarin Forms [Електронний ресурс] / Metanit. – Режим доступу до ресурсу: https://metanit.com/sharp/xamarin/.

17. Windows Presentation Foundation documentation [Електронний ресурс] / MSDN. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-5.0.

18. Xamarin documentation [Електронний ресурс] / MSDN. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/xamarin/.

19. C# documentation [Електронний ресурс] / MSDN. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/dotnet/csharp/.

ДОДАТОК А

(обов’язковий)

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ

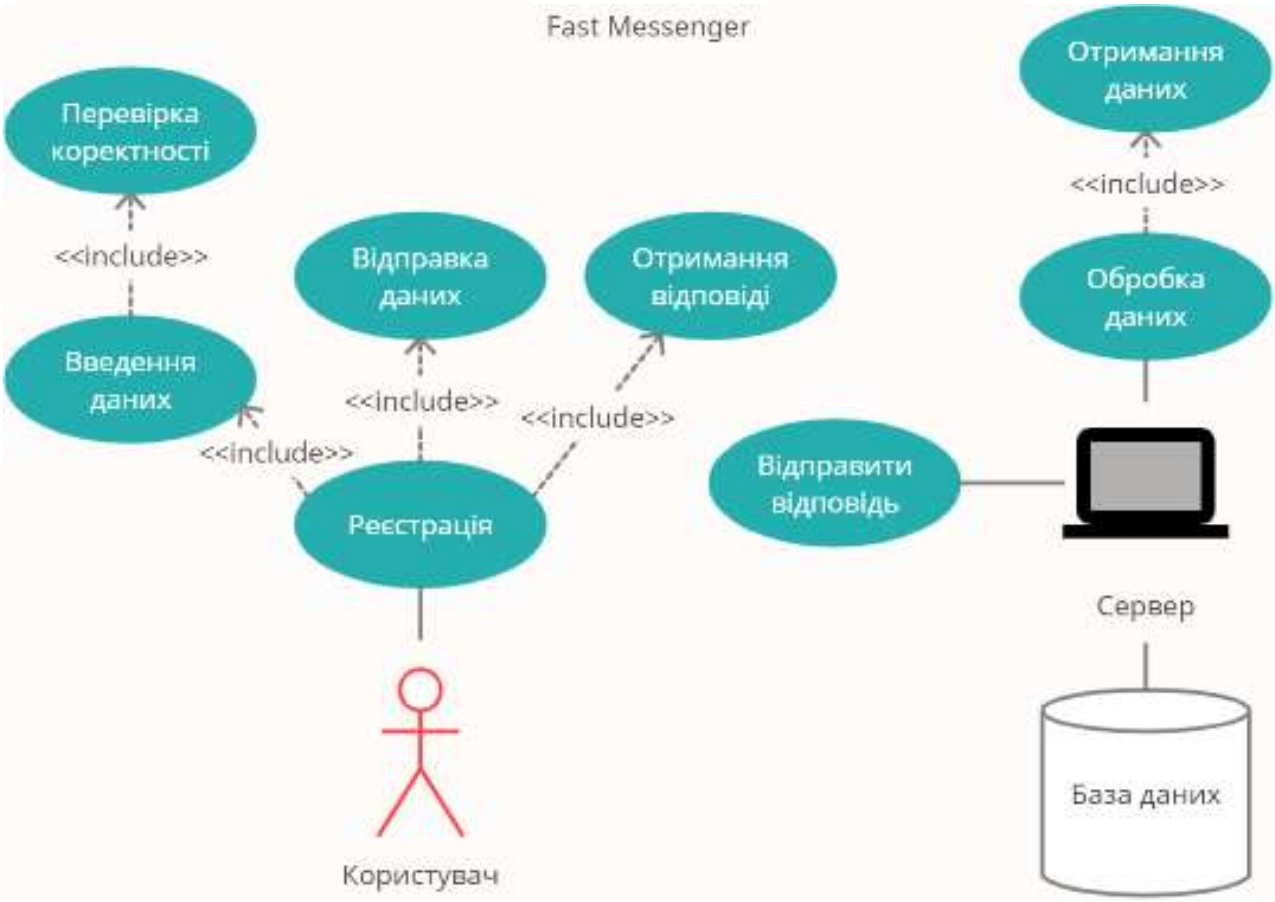


Рисунок А.1 – Діаграма використання реєстрації

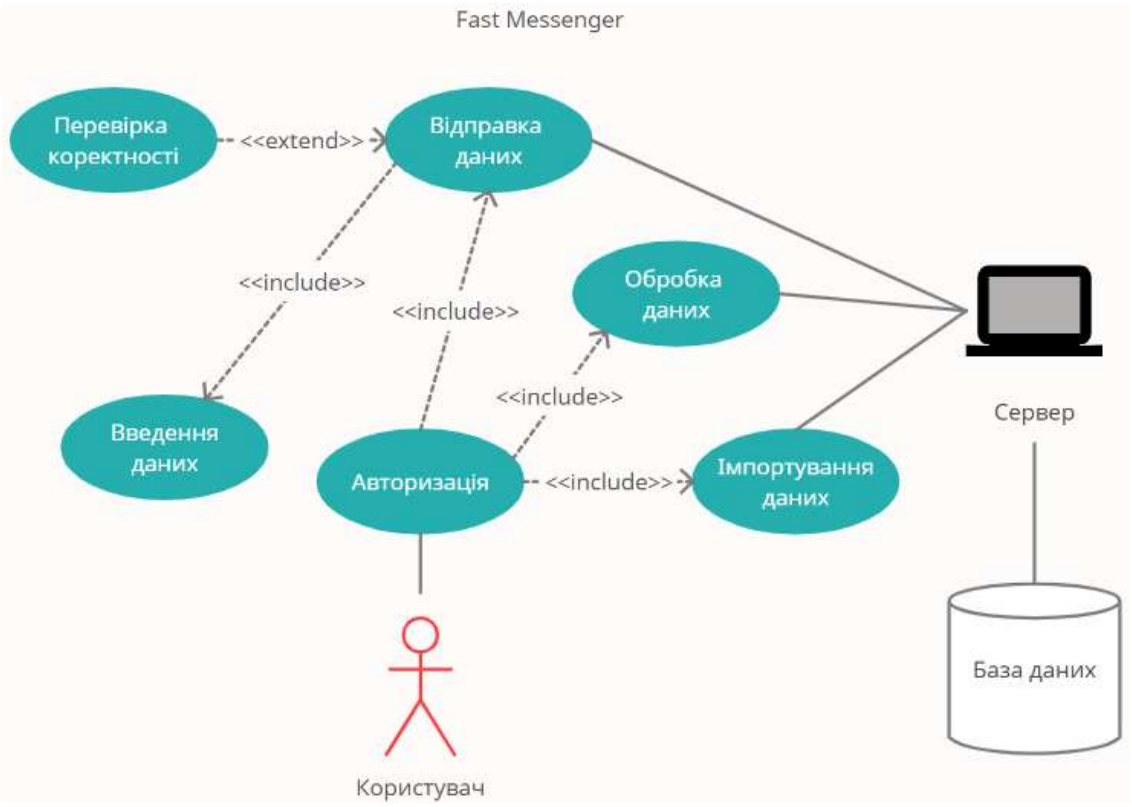


Рисунок А.2 – Діаграма використання авторизації

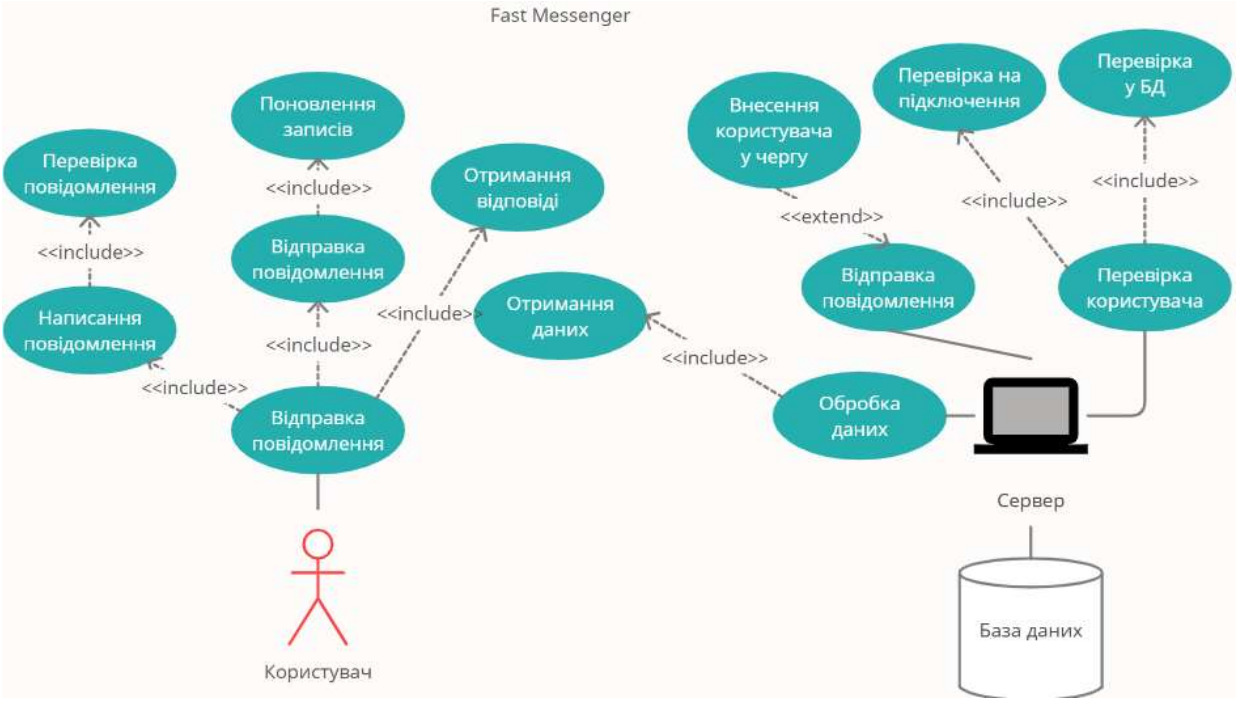


Рисунок А.3 – Діаграма варіантів відправки повідомлення

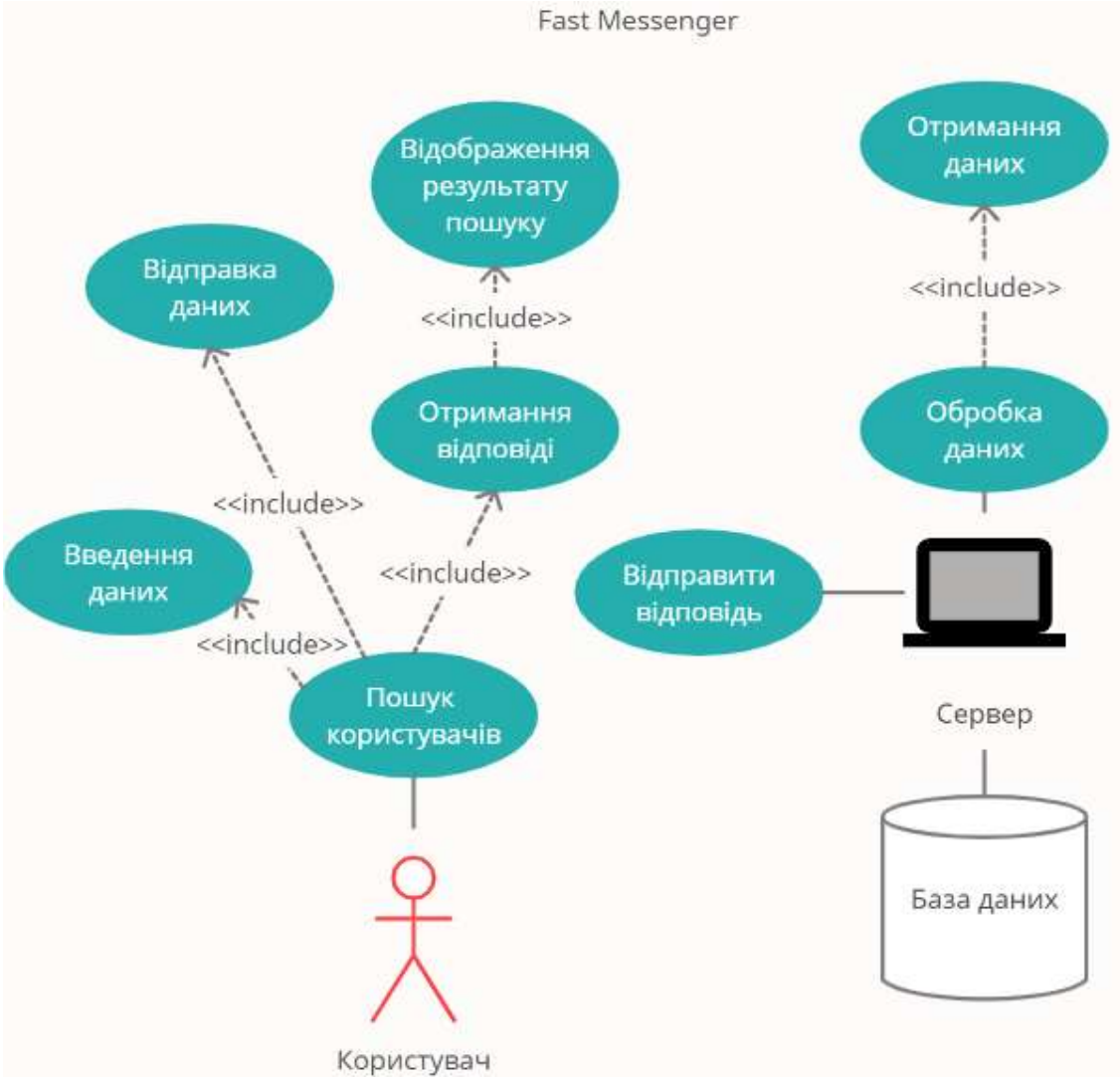


Рисунок А.4 – Діаграма варіантів пошуку користувачів

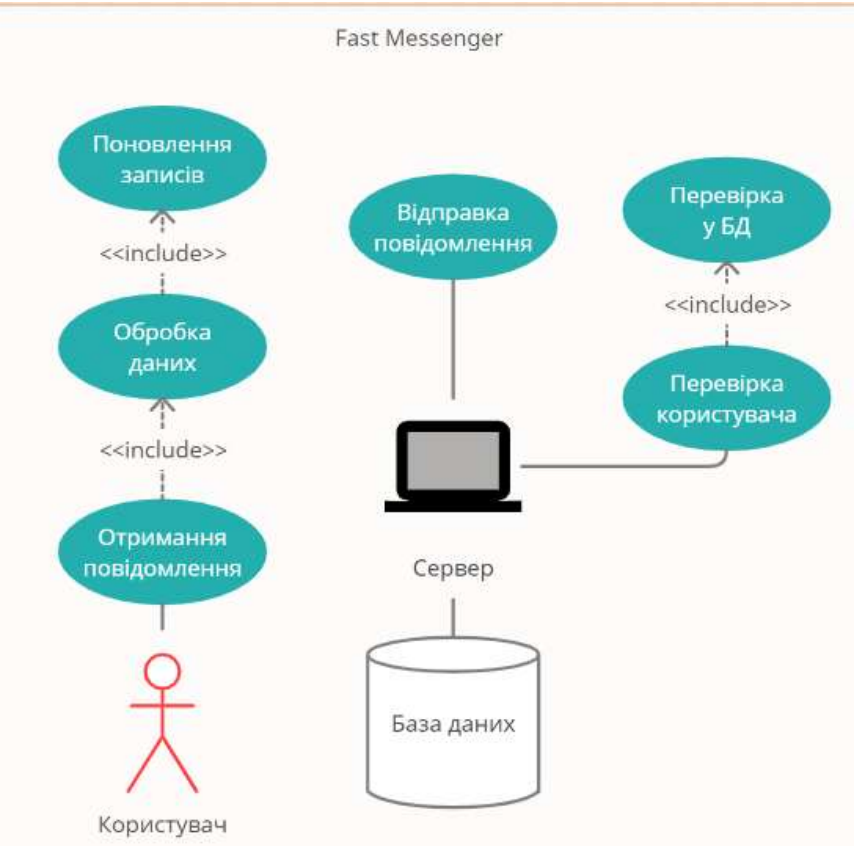


Рисунок А.5 – Діаграма використання отримки повідомлення

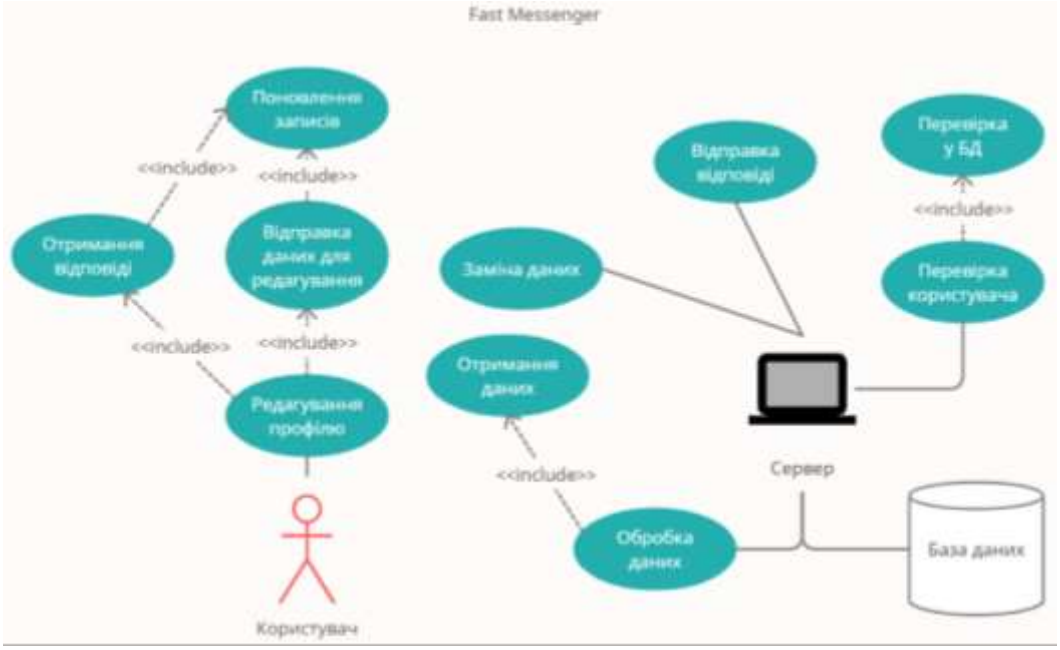


Рисунок А.6 – Діаграма використання редагування профілю

ДОДАТОК Б

(обов'язковий)

ДІАГРАМА КЛАСІВ

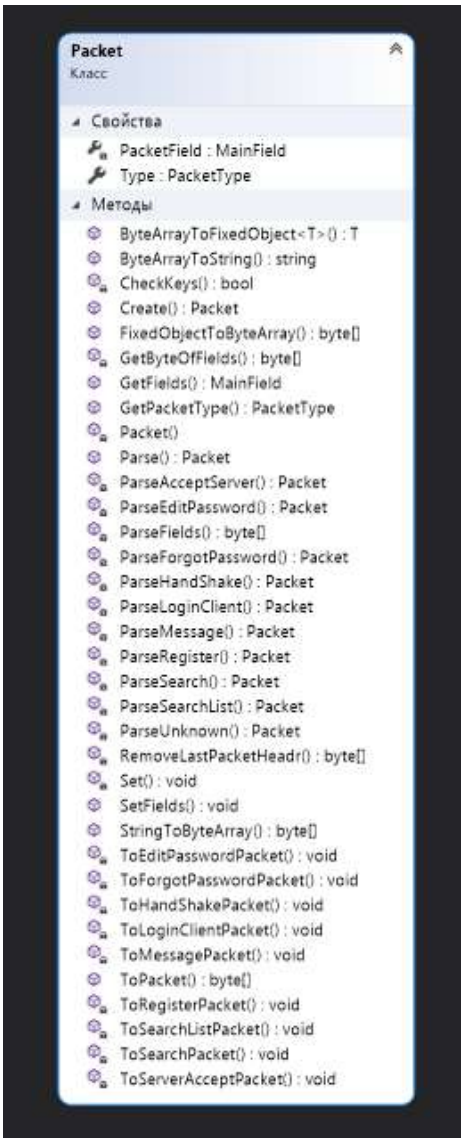


Рисунок Б.1 – Діаграма класу Packet

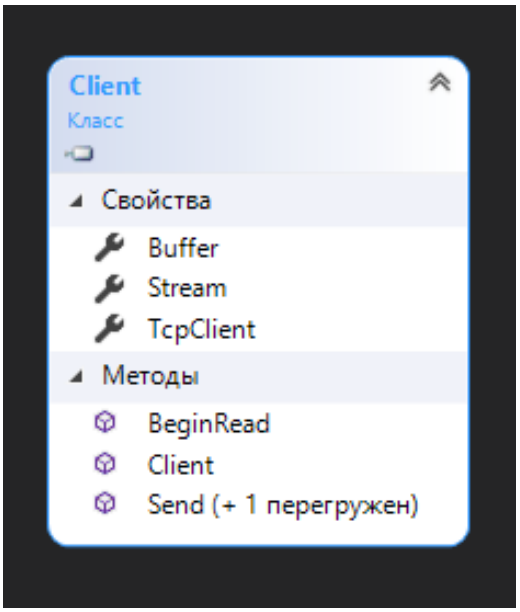


Рисунок Б.2 – Діаграма класу Client

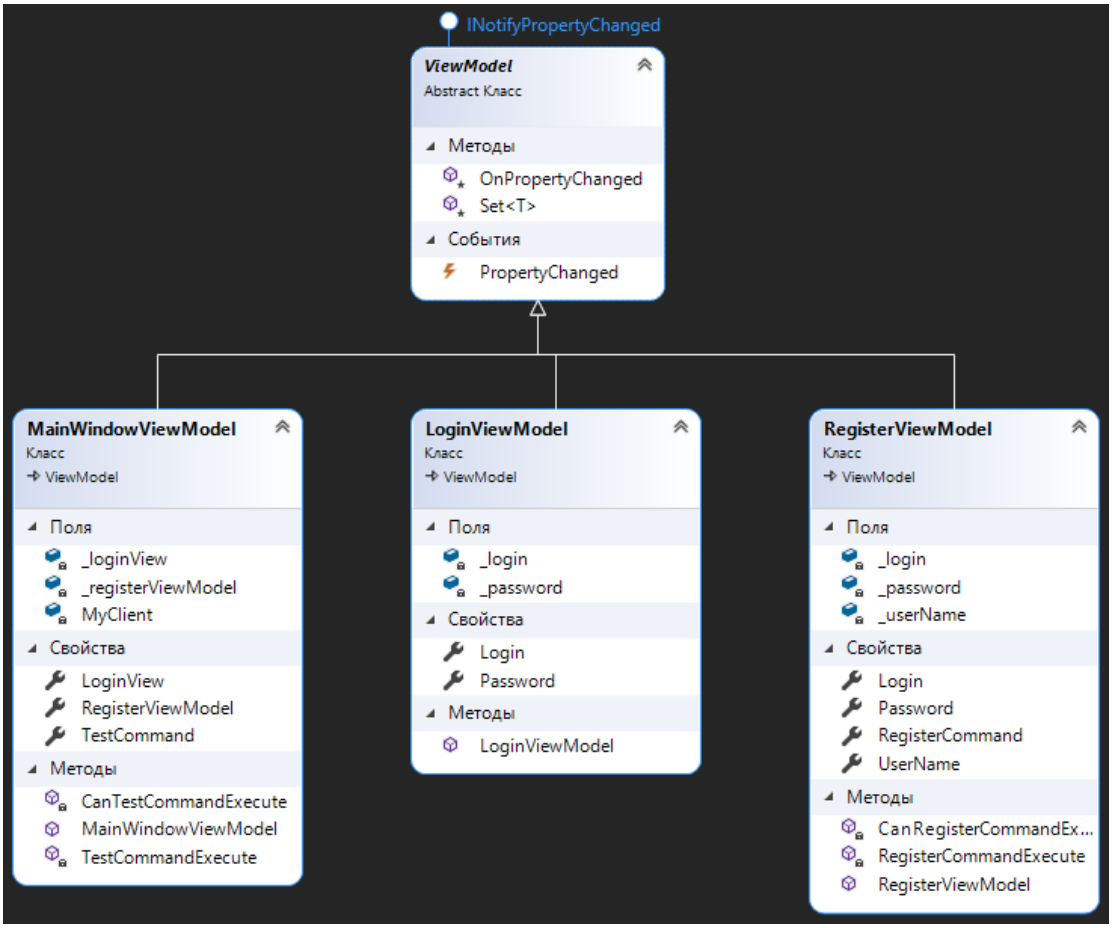


Рисунок Б.3 – Діаграма класів ViewModel

ДОДАТОК В

(обов'язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

В.1 Програмний код базового класу ViewModel

```
    internal abstract class ViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        protected virtual void OnPropertyChanged(string propertyName = null)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }

        protected virtual bool Set<T>(ref T field, T value, string propertyName
= null)
        {
            if (Equals(field, value))
            {
                return false;
            }

            field = value;
            OnPropertyChanged(propertyName);
            return true;
        }
    }
```

В.2 Програмний код базового класу Command для WPF додатку

```
    internal abstract class Command : ICommand
    {
        public event EventHandler CanExecuteChanged
        {
            add => CommandManager.RequerySuggested += value;
            remove => CommandManager.RequerySuggested -= value;
        }

        public abstract bool CanExecute(object parameter);

        public abstract void Execute(object parameter);
    }
```

В.3 Програмний код класу реєстрації команд для WPF додатку

```
    internal class RelayCommand : Command
    {
        private readonly Action<object> _execute;
```

```
        private readonly Func<object, bool> _canExecute;

        public RelayCommand(Action<object> execute, Func<object, bool>
canExecute = null)
        {
            _execute = execute;
            _canExecute = canExecute;
        }

        public override bool CanExecute(object parameter)
        {
            return _canExecute?.Invoke(parameter) ?? true;
        }

        public override void Execute(object parameter)
        {
            _execute(parameter);
        }
    }
```

В.4 Програмний код базового класу Command для Xamarin додатку

```
    internal class RelayCommand : Command
    {
        public RelayCommand(Action<object> execute) : base(execute) { }

        public RelayCommand(Action execute) : this(e => execute()) { }

        public RelayCommand(Action<object> execute,
                        Func<object, bool> canExecute,
                        INotifyPropertyChanged propertyChanged = null)
            : base(execute, canExecute)
        {
            if (propertyChanged != null)
            {
                propertyChanged.PropertyChanged += delegate {
ChangeCanExecute(); };
            }
        }

        public RelayCommand(Action execute,
                        Func<bool> canExecute,
                        INotifyPropertyChanged propertyChanged = null)
            : this(e => execute(), e => canExecute(), propertyChanged) { }
    }
```

В.5 Програмний код методу для заповнення контейнера пакету

```
      public static void Set(List<byte> packet, byte[] field)
       {
            for (int i = 0; i < field.Length; i++)
            {
                packet.Add(field[i]);
            }
       }
```

В.6 Програмний код методу реалізація методу ToPacket класу Packet

```
      public byte[] ToPacket()
       {
            Aes aes = new Aes();
            var packet = new List<byte>();
            Set(packet, new byte[] { 0xAF, 0xAA, 0xAF, 0xAF, 0xAA, 0xAF,
(byte)Type });

            if(PacketField != null)
            {
                PacketField.ToPacketField(packet);
            }

            Set(packet, new byte[] { 0xFF, 0x00, 0xFF, 0x00, 0xFF });

            return aes.EncryptToByte(packet.ToArray());
       }
```

В.7 Програмний код методу реалізація методу Parse класу Packet

```
      public static Packet Parse(byte[] packet)
       {
            Aes aes = new Aes();
            var packetClipped = GetByteOfFields(packet);

            var decrypt = aes.Decrypt(packetClipped);

            if (decrypt[0] != 0xAF || decrypt[1] != 0xAA || decrypt[2] != 0xAF
|| decrypt[3] != 0xAF || decrypt[4] != 0xAA || decrypt[5] != 0xAF)
            {
                return ParseUnknown();
            }

            if (decrypt[decrypt.Length-1] == 0xFF && decrypt[decrypt.Length - 2]
== 0x00 && decrypt[decrypt.Length - 3] == 0xFF && decrypt[decrypt.Length - 4] ==
0x00 && decrypt[decrypt.Length - 5] == 0xFF)
            {
                decrypt = RemoveLastPacketHeadr(decrypt);
            }
            else
            {
                return ParseUnknown();
            }

            var type = decrypt[6];
            var EMTPPacket = Create((PacketType)type);
            var field = decrypt.Skip(7).ToArray();
            EMTPPacket.PacketField =
PacketManager.GetPacketFieldType(EMTPPacket.Type);

            if (PacketType.Unknown == EMTPPacket.Type)
            {
                return ParseUnknown();
            }

            return EMTPPacket.PacketField.Parse(field, EMTPPacket);
        }
```

В.8 Програмний код класу, який відповідає за обмін даними

```
public class Client
    {
        public byte[] Buffer { get; set; }

        public TcpClient TcpClient { get; set; }

        public NetworkStream Stream { get; set; }

        private EventWaitHandle Handle { get; set; }

        private Task SendPacket { get; set; }

        public Client()
        {
            Handle = new EventWaitHandle(false, EventResetMode.AutoReset);
        }

        public void BeginRead()
        {
            Stream.BeginRead(Buffer, 0, Buffer.Length, new
AsyncCallback(EndRead), false);
            Handle.WaitOne();
        }

        private void EndRead(IAsyncResult result)
        {
            int bytes = 0;
            if (TcpClient.Connected)
            {
                try
                {
                    bytes = Stream.EndRead(result);
                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.Message);
                }
            }

            Handle.Set();
        }

        public void Send(string message)
        {
            if (SendPacket == null || SendPacket.IsCompleted)
            {
                SendPacket = Task.Factory.StartNew(() => BeginWrite(message));
            }
            else
            {
                SendPacket.ContinueWith(i => BeginWrite(message));
            }
        }

        public void Send(byte[] packet)
        {
            if (SendPacket == null || SendPacket.IsCompleted)
```

```
            {
                SendPacket = Task.Factory.StartNew(() => BeginWrite(packet));
            }
            else
            {
                SendPacket.ContinueWith(i => BeginWrite(packet));
            }
        }

        private void BeginWrite(byte[] packet)
        {
            if (TcpClient.Connected)
            {
                Stream.BeginWrite(packet, 0, packet.Length, new
AsyncCallback(EndWrite), null);
            }
        }

        private void BeginWrite(string message)
        {
            var buffer = Encoding.UTF8.GetBytes(message);

            if (TcpClient.Connected)
            {
                Stream.BeginWrite(buffer, 0, buffer.Length, new
AsyncCallback(EndWrite), null);
            }
        }

        private void EndWrite(IAsyncResult result)
        {
            if (TcpClient.Connected)
            {
                Stream.EndWrite(result);
            }
        }
    }
```

В.9 Програмний код розмітки навігації форм авторизації мобільного додатку

```
<Shell xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="FastMessanger.Views.AppShell"
             xmlns:local="clr-
namespace:FastMessanger.ViewModels.AuthorizeViewModels"
             xmlns:pages="clr-namespace:FastMessanger.Views.AuthorizeView"
             Shell.NavBarIsVisible="False">


    <TabBar Title="Login">
        <ShellContent ContentTemplate="{DataTemplate pages:LoginPage}"
Route="loginpage" Title="Login"/>
        <Tab Title="Register">
            <ShellContent ContentTemplate="{DataTemplate pages:RegisterPage}"
Route="registerpage"/>
        </Tab>
        <Tab Title="Forgot password">
```

```
            <ShellContent ContentTemplate="{DataTemplate
pages:ForgotPasswordPage}" Route="forgotpasswordpage"/>
        </Tab>
    </TabBar>
    <FlyoutItem IsVisible="False" IsEnabled="False">
        <ShellContent ContentTemplate="{DataTemplate pages:UpdatePasswordPage}"
Route="updatepasswordpage"/>
    </FlyoutItem>
    <FlyoutItem IsVisible="False" IsEnabled="False">
        <ShellContent ContentTemplate="{DataTemplate pages:WriteSecretWordPage}"
Route="writesecretwordpage"/>
    </FlyoutItem>

</Shell>
```

В.10 Програмний код розмітки форми авторизації мобільного додатку

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:vm="clr-
namespace:FastMessanger.ViewModels.AuthorizeViewModels"
             x:Class="FastMessanger.Views.AuthorizeView.LoginPage"
             Shell.NavBarIsVisible="False">

    <ContentPage.BindingContext>
        <vm:LoginViewModel/>
    </ContentPage.BindingContext>

    <ContentPage.Resources>
        <Style TargetType="Label">
            <Setter Property="FontSize" Value="16"/>
            <Setter Property="Margin" Value="5"/>
        </Style>

        <Style TargetType="Entry">
            <Setter Property="FontSize" Value="16"/>
            <Setter Property="TextColor" Value="Black"/>
        </Style>

        <Style x:Key="helpButton" TargetType="Button">
            <Setter Property="Background" Value="WhiteSmoke"/>
            <Setter Property="TextColor" Value="Green"/>
            <Setter Property="Opacity" Value="10"/>
            <Setter Property="Margin" Value="5"/>
        </Style>
    </ContentPage.Resources>

    <ContentPage.Content>
        <StackLayout Margin="10" Orientation="Vertical"
VerticalOptions="Center">
            <Label Text="Login"/>
            <Entry Text="{Binding Login, Mode=TwoWay}"/>

            <Label Text="Password"/>
            <Entry IsPassword="True" Text="{Binding Password, Mode=TwoWay}"/>

            <Button Text="Authorize"
                    Background="Green"
```

```
                    TextColor="White"
                    FontSize="16"
                    CornerRadius="15"
                    Command="{Binding AuthorizeCommand}"/>
        </StackLayout>
    </ContentPage.Content>

</ContentPage>
```

В.11 Програмний код розмітки представлення реєстрації для мобільного додатку

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:vm="clr-
namespace:FastMessanger.ViewModels.AuthorizeViewModels"
             x:Class="FastMessanger.Views.AuthorizeView.RegisterPage"
             Shell.NavBarIsVisible="False">

    <ContentPage.BindingContext>
        <vm:RegisterViewModel/>
    </ContentPage.BindingContext>

    <ContentPage.Resources>
        <Style TargetType="Label">
            <Setter Property="FontSize" Value="16"/>
            <Setter Property="Margin" Value="5"/>
        </Style>
        <Style TargetType="Entry">
            <Setter Property="FontSize" Value="16"/>
            <Setter Property="TextColor" Value="Black"/>
        </Style>
    </ContentPage.Resources>

    <ContentPage.Content>
        <StackLayout Margin="10" Orientation="Vertical"
VerticalOptions="Center">

            <Label Text="Login"/>
            <Entry Text="{Binding Login, Mode=TwoWay}"/>

            <Label Text="User name"/>
            <Entry Text="{Binding UserName, Mode=TwoWay}"/>

            <Label Text="Password"/>
            <Entry IsPassword="True" Text="{Binding Password, Mode=TwoWay}"/>

            <Label Text="Confirm password"/>
            <Entry IsPassword="True" Text="{Binding ConfirmPassword,
Mode=TwoWay}"/>

            <Button Text="Register"
                    Background="Green"
                    TextColor="White"
                    FontSize="16"
                    Margin="10"
                    CornerRadius="15"
                    Command="{Binding RegisterComamnd}"/>
```

```
            <Label VerticalOptions="Center"
                   HorizontalOptions="Center"
                   Text="{Binding Info, Mode=TwoWay}"
                   TextColor="Red"/>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

В.12 Програмний код моделі представлення для сторінки авторизації мобільного додатку

```
    internal class RegisterViewModel : ViewModel
    {
        private string _info;

        public string Info
        {
            get
            {
                return _info;
            }
            set
            {
                Set(ref _info, value, nameof(Info));
            }
        }

        private string _login;

        public string Login
        {
            get
            {
                return _login;
            }
            set
            {
                Set(ref _login, value, nameof(Login));
            }
        }

        private string _userName;

        public string UserName
        {
            get
            {
                return _userName;
            }
            set
            {
                Set(ref _userName, value, nameof(UserName));
            }
        }

        private string _password;

        public string Password
```

```
        {
            get
            {
                return _password;
            }
            set
            {
                Set(ref _password, value, nameof(Password));
            }
        }

        private string _confirmPassword;

        public string ConfirmPassword
        {
            get
            {
                return _confirmPassword;
            }
            set
            {
                Set(ref _confirmPassword, value, nameof(ConfirmPassword));
            }
        }

        public Command RegisterComamnd
        {
            get
            {
                return new RelayCommand(RegisterComamndExecute,
CanRegisterComamndExecute);
            }
        }

        private async void RegisterComamndExecute(object obj)
        {
            var content = Content.GetContent();
            if(CheckIsEntryCorrect())
            {
                content._registerField = new RegisterFields() { Login = Login,
UserName = UserName, Password = Password };
                Register = new Register(Login, UserName, Password);

                if(TryRegister())
                {
                    await Shell.Current.GoToAsync("//writesecretwordpage");
                }
                else
                {
                    Info = "Has some problem, try again";
                }
            }
            else
            {
                Info = "Confirm password false";
            }
        }

        private bool CanRegisterComamndExecute(object obj)
        {
```

```
            return true;
        }

        private bool CheckIsEntryCorrect()
        {
            return Password == ConfirmPassword;
        }

        public RegisterViewModel()
        {

        }

      internal bool TryRegister()
        {
            obj = new Client();
            obj.TcpClient = new TcpClient();
            obj.TcpClient.Connect(IPAddress.Parse("192.168.0.103"), 8888);
            obj.Stream = obj.TcpClient.GetStream();
            obj.Buffer = new byte[obj.TcpClient.ReceiveBufferSize];

            if(secretWord == null)
            {
                return RegisterWithoutSecret(obj);
            }
            else
            {
                return CompleteRegister(obj);
            }
        }

        private bool CompleteRegister(Client obj)
        {
            var registerPacket = Packet.Create(PacketType.Register);
            registerPacket.SetFields(new RegisterFields() { Login = login,
Password = password, UserName = userName, SecretWord = secretWord });
            var packetBytes = registerPacket.ToPacket();
            obj.Send(packetBytes);

            if (obj.TcpClient.Connected)
            {
                try
                {
                    obj.BeginRead();
                    if (obj.Buffer.Length != 0)
                    {
                        var acceptPacket = Packet.Parse(obj.Buffer);
                        if (acceptPacket.Type == PacketType.AcceptServer)
                        {
                            var field =
(AcceptServerFields)acceptPacket.GetFields();
                            if (field.Answer == SERVERACCEPTWORD)
                            {
                                obj.Buffer = new byte[65536];
                                obj.TcpClient.Client.Close();
                                obj.TcpClient.Close();
                                return true;
                            }
                        }
                    }
```

```
                }
                catch
                {
                    return false;
                }
            }

            return false;
        }

        private bool RegisterWithoutSecret(Client obj)
        {
            var registerPacket = Packet.Create(PacketType.Register);
            registerPacket.SetFields(new RegisterFields() { Login = login,
Password = password, UserName = userName });
            var packetBytes = registerPacket.ToPacket();
            obj.Send(packetBytes);

            if (obj.TcpClient.Connected)
            {
                try
                {
                    obj.BeginRead();
                    if (obj.Buffer.Length != 0)
                    {
                        var acceptPacket = Packet.Parse(obj.Buffer);
                        if (acceptPacket.Type == PacketType.AcceptServer)
                        {
                            var field =
(AcceptServerFields)acceptPacket.GetFields();
                            if (field.Answer == "DAROVA")
                            {
                                obj.Buffer = new byte[65536];
                                obj.TcpClient.Client.Close();
                                obj.TcpClient.Close();
                                return true;
                            }
                        }
                    }
                }
                catch
                {
                    return false;
                }
            }

            return false;
        }
    }
```

В.13 Програмний код розмітки навігації форм акаунта мобільного додатку

```
<Shell xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="FastMessanger.Views.MainAppShell"
             xmlns:pages="clr-namespace:FastMessanger.Views.MainViews"
       xmlns:vm="clr-namespace:FastMessanger.ViewModels.MainPagesViewModels"
       Shell.NavBarIsVisible="False">
```

```
    <Shell.BindingContext>
        <vm:MainShellViewModel/>
    </Shell.BindingContext>

    <Shell.FlyoutHeader>
        <StackLayout Background="White" Padding="10">
            <Label Text="Fast Messanger" HorizontalOptions="Center"
VerticalOptions="Center"/>
        </StackLayout>
    </Shell.FlyoutHeader>

    <FlyoutItem Title="Chats">
        <ShellContent ContentTemplate="{DataTemplate pages:ChatsViews}"
Route="chatsviews" Title="Chats"/>
        <Tab Title="Search">
            <ShellContent ContentTemplate="{DataTemplate pages:SearchUserPage}"
Route="searchuserpage"/>
        </Tab>
    </FlyoutItem>
    <FlyoutItem Title="Account">
        <ShellContent ContentTemplate="{DataTemplate pages:AccountPage}"
Route="accountpage"/>
    </FlyoutItem>

    <MenuItem Text="Logout" Command="{Binding LogoutCommand}"/>

                                    </Shell>
```

В.14 Програмний код методу менеджера пакетів клієнта

```
      private void ClientTypeManager(Client client)
        {
            var acceptPacket = Packet.Parse(client.Buffer);
            if (acceptPacket.Type == PacketType.HandShake)
            {
                HandleHandShake(acceptPacket);
                return;
            }

            if (acceptPacket.Type == PacketType.Message)
            {
                HandleMessage(acceptPacket);
                return;
            }

            if (acceptPacket.Type == PacketType.SearchList)
            {
                HandleSearchList(acceptPacket);
                return;
            }

            if (acceptPacket.Type == PacketType.Unknown)
            {
                return;
            }
        }
```

В.15 Програмний код форми авторизації для комп'ютерного додатку

```
<Page x:Class="Client.Pages.Authorize.LoginPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:vm="clr-namespace:Client.ViewModels.PageViewModel"
xmlns:CustomCountrols="clr-namespace:Client.CustomCountrols"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="LoginPage"
      Background="White"
      Style="{StaticResource {x:Type Page}}">

    <Page.DataContext>
        <vm:LoginViewModel/>
    </Page.DataContext>

    <Page.Resources>
        <Style x:Key="primarybutton" TargetType="{x:Type Button}">
            <Setter Property="MinWidth" Value="120"/>
            <Setter Property="VerticalAlignment" Value="Center"/>
            <Setter Property="Height" Value="40"/>
            <Setter Property="FontWeight" Value="SemiBold"/>
            <Setter Property="Background" Value="#FF01A3FF"/>
            <Setter Property="Foreground" Value="White"/>
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="{x:Type Button}">
                        <Border x:Name="bd"
                                Background="{TemplateBinding Background}"
                                BorderThickness="{TemplateBinding
BorderThickness}"
                                CornerRadius="10">
                            <ContentPresenter
                                HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}"
                                VerticalAlignment="{TemplateBinding
VerticalContentAlignment}"
                                TextBlock.Foreground="{TemplateBinding
Foreground}"/>
                        </Border>
                        <ControlTemplate.Triggers>
                            <Trigger Property="IsMouseOver" Value="True">
                                <Setter TargetName="bd" Property="Background"
Value="#80d0FF"/>
                            </Trigger>
                        </ControlTemplate.Triggers>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
    </Page.Resources>

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="70"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <!--Title-->
```

```
        <TextBlock
          VerticalAlignment="Center"
          HorizontalAlignment="Left"
          Margin="35,0"
          FontSize="30"
          Opacity="0">
            <TextBlock.Style>
                <Style>
                    <Style.Triggers>
                        <EventTrigger RoutedEvent="FrameworkElement.Loaded">
                            <BeginStoryboard>
                                <Storyboard SpeedRatio="1">
                                    <DoubleAnimation
Storyboard.TargetProperty="Opacity" To="1"/>
                                </Storyboard>
                            </BeginStoryboard>
                        </EventTrigger>
                    </Style.Triggers>
                </Style>
            </TextBlock.Style>
            <Run
                Text="Fast Messenger"/>
            <Run
                FontWeight="Bold"
                Text="Login"/>
        </TextBlock>

        <!--Login UI-->
        <Grid Grid.Row="1"
            Width="400"
            HorizontalAlignment="Center"
            VerticalAlignment="Center">
            <Grid.RowDefinitions>
                <RowDefinition Height="100"/>
                <!--User Login-->
                <RowDefinition Height="100"/>
                <!--Password-->
                <RowDefinition Height="100"/>
                <RowDefinition/>
                <!--Button-->
                <RowDefinition Height="100"/>
                <!--Note-->
                <RowDefinition/>
            </Grid.RowDefinitions>

            <TextBlock
                    VerticalAlignment="Center"
                    FontSize="30">
                <Run
                    FontWeight="SemiBold"
                    Text="Hello, "/>
                <Run
                    Text="User"
                    Foreground="#FF01A3ff"
                    FontWeight="Bold"
                    x:Name="userName"/>
            </TextBlock>

            <!--Custom login textbox-->
            <CustomCountrols:TextBoxWithPlaceHolder
```

```
        Grid.Row="1"
        PlaceHolder="Login"
        DataContext="{Binding Path=LoginView.Login}"/>

<!--Password textBox-->
<CustomCountrols:TextBoxWithPlaceHolder
        Grid.Row="2"
        PlaceHolder="Password"
        IsPassword="True"
        DataContext="{Binding Path=LoginView.Password}"/>

<Grid
        Grid.Row="3"
        Margin="10,0"
        Height="60">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="100"/>
        <ColumnDefinition/>
        <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>

    <!--Register-->
    <Button
        FontWeight="SemiBold"
        VerticalAlignment="Center"
        MaxWidth="135"
        FontSize="15"
        Background="WhiteSmoke"
        BorderThickness="0"
        Style="{StaticResource {x:Static ToolBar.ButtonStyleKey}}"
        Content="Register now">
    </Button>

    <!--Forgot Password-->
    <Button
        FontWeight="SemiBold"
        VerticalAlignment="Center"
        MaxWidth="135"
        FontSize="15"
        Background="WhiteSmoke"
        BorderThickness="0"
        Grid.Column="1"
        Style="{StaticResource {x:Static ToolBar.ButtonStyleKey}}"
        Content="Forgot Password?"/>

    <!--Login Button-->
    <Button
        Grid.Column="2"
        Content="Login"
        Command="{Binding Path=TestCommand}"
        Style="{StaticResource primarybutton}"/>
</Grid>

<!--note-->
<TextBlock
    Grid.Row="4"
    VerticalAlignment="Bottom"
    TextWrapping="Wrap"
    TextAlignment="Center"
    Foreground="#A3A3A4"
```

```
                FontWeight="SemiBold">
                <Run
                    Text="Note:"
                    FontWeight="Bold"/>
                <LineBreak/>
                <Run
                    Text="Do not disclose your login, password and verification
code to anyone. We will never ask for your credentials."/>
            </TextBlock>
        </Grid>
    </Grid>

</Page>
```

В.16 Програмний код модуля головного вікна

```
internal class ChatsListViewModel : ViewModel
{
    private string _userMessage = "";

    public string UserMessage
    {
        get
        {
            return _userMessage;
        }
        set
        {
            Set(ref _userMessage, value, nameof(UserMessage));
        }
    }

    private string _message;

    public string Message
    {
        get
        {
            return _message;
        }
        set
        {
            Set(ref _message, value, nameof(Message));
        }
    }

    private ObservableCollection<Message> _messages;

    public ObservableCollection<Message> Messages
    {
        get
        {
            return _messages;
        }
        set
        {
            Set(ref _messages, value, nameof(Messages));
        }
    }
```

```
private string _searchInfo;

public string SearchInfo
{
    get
    {
        return _searchInfo;
    }
    set
    {
        Set(ref _searchInfo, value, nameof(SearchInfo));
    }
}

private Chat _chat;

public Chat Chat
{
    get
    {
        return _chat;
    }
    set
    {
        Set(ref _chat, value, nameof(Chat));
    }
}

private ObservableCollection<Chat> _chatsList;

public ObservableCollection<Chat> ChatsList
{
    get
    {
        return _chatsList;
    }
    set
    {
        Set(ref _chatsList, value, nameof(ChatsList));
    }
}

private SearchUser _searchUser;

public SearchUser SearchUser
{
    get
    {
        return _searchUser;
    }
    set
    {
        Set(ref _searchUser, value, nameof(SearchUser));
    }
}

private string _searchUserText;

public string SearchUserText
{
```

```
            get
            {
                return _searchUserText;
            }
            set
            {
                Set(ref _searchUserText, value, nameof(SearchUserText));
            }
        }

        public ICommand GoToThisChatCommand
        {
            get
            {
                return new RelayCommand(GoToThisChatCommandExecute,
CanGoToThisChatCommandExecute);
            }
        }

        private void GoToThisChatCommandExecute(object obj)
        {
            RewriteCollection(_chat.Messages);
        }

        private void RewriteCollection(ObservableCollection<Message> collection)
        {
            _messages.Clear();

            for(int i = 0; i < collection.Count; i++)
            {
                _messages.Add(collection[i]);
            }
        }

        private bool CanGoToThisChatCommandExecute(object obj)
        {
            return Chat != null;
        }

        public ICommand SettingCommand
        {
            get
            {
                return new RelayCommand(SettingCommandExecute);
            }
        }

        private MainWindow mainWindow3 { get => Application.Current.MainWindow
as MainWindow; }

        private void SettingCommandExecute(object obj)
        {
            mainWindow3.mainFrame.Navigate(new
Uri("/pages/mainpages/settingpage.xaml", UriKind.RelativeOrAbsolute));
        }

        private MainWindow mainWindow2 { get => Application.Current.MainWindow
as MainWindow; }

        public ICommand LogoutCommand
```

```
        {
            get
            {
                return new RelayCommand(LogoutCommandExecute);
            }
        }

        private void LogoutCommandExecute(object obj)
        {
            XmlSerializer formatter = new XmlSerializer(typeof(Chat));
            using(FileStream stream = new FileStream("thisChat.xml",
FileMode.OpenOrCreate))
            {
                formatter.Serialize(stream, Chat);
            }

            client.TcpClient.Close();
            client.TcpClient.Client.Close();
            client.Buffer = new byte[65526];

            mainWindow2.mainFrame.Navigate(new
Uri("/pages/authorize/loginpage.xaml", UriKind.RelativeOrAbsolute));
        }

        public ICommand SendMessageCommand
        {
            get
            {
                return new RelayCommand(SendMessageCommandExecute,
CanSendMessageCommandExecute);
            }
        }

        private void SendMessageCommandExecute(object obj)
        {
            var content = Content.GetContent();

            var packet = Packet.Create(PacketType.Message);
            packet.SetFields(new MessageFields()
            {
                Message = _userMessage,
                Login = _chat.Login,
                UserName = _chat.UserName,
                WhoSendLogin = content.Login,
                WhoSendUserName = "ahahha"
            });

            Chat.Messages.Add(new Message() { UserMessage = UserMessage, WhoSend
= "You"});
            Chat.LastMessage = _userMessage;
            _messages.Add(new Message() { UserMessage = UserMessage, WhoSend =
"You" });
            UserMessage = string.Empty;

            XmlSerializer formatter = new XmlSerializer(typeof(Chat));
            using (FileStream stream = new FileStream("thisChat.xml",
FileMode.OpenOrCreate))
            {
                formatter.Serialize(stream, Chat);
            }
```

```
            client.Send(packet.ToPacket());
        }

        private bool CanSendMessageCommandExecute(object obj)
        {
            return _userMessage.Length > 0;
        }

        public ICommand AddChatCommand
        {
            get
            {
                return new RelayCommand(AddChatCommandExecute,
CanAddChatCommandExecute);
            }
        }

        private void AddChatCommandExecute(object obj)
        {
            _chatsList.Add(new Chat() { UserName = _searchUser.UserName, Login =
_searchUser.Login, Messages = new ObservableCollection<Message>() });
            _searchInfo = string.Empty;
            SearchUserText = string.Empty;
            _searchUser = null;
        }

        private bool CanAddChatCommandExecute(object obj)
        {
            return SearchUser != null;
        }

        public ICommand SearchUserCommand
        {
            get
            {
                return new RelayCommand(SearchUserCommandExecute,
CanSearchUserCommandExecute);
            }
        }
        private void SearchUserCommandExecute(object obj)
        {
            var packet = Packet.Create(PacketType.Search);
            packet.SetFields(new SearchFields() { UserName = _searchUserText });
            var packetBytes = packet.ToPacket();

            client.Send(packetBytes);
        }
        private bool CanSearchUserCommandExecute(object obj)
        {
            return true;
        }

        private EMTP.Client client;

        public ChatsListViewModel()
        {
            _chatsList = new ObservableCollection<Chat>();
            BindingOperations.EnableCollectionSynchronization(_chatsList, new
object());
```

```
            _messages = new ObservableCollection<Message>();
            BindingOperations.EnableCollectionSynchronization(_messages, new
object());

            if(File.Exists("thisChat.xml"))
            {
                XmlSerializer formatter = new XmlSerializer(typeof(Chat));
                using (var stream = new FileStream("thisChat.xml",
FileMode.OpenOrCreate))
                {
                    var newChat = (Chat)formatter.Deserialize(stream);
                    _chatsList.Add(newChat);
                }
            }

            var content = Content.GetContent();
            client = content.client;
            Start();
        }
```

ДОДАТОК Г

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проекту програмної системи обміну повідомленнями “FastMessanger”.

1 Найменування та область застосування

Найменування програмна система обміну повідомленнями з використанням стійких алгоритмів шифрування. Данний програмний продукт розробляється для створення безпеченої системи обміну повідомленнями

2 Підстави для розробки

Підставою для розробки є “Завдання на дипломний проект”, затверджене завідуючим кафедри інженерії програмного забезпечення.

3 Призначення розробки

Функціональне призначення: безпечено відправляти повідомлення ішним користувачам, реєстрація і авторизація.

Експлуатаційне призначення: встановити додатки можна на будь-який комп’ютер і операційну систему, додаток працюватиме на мобільних пристроях Android і IOS.

4 Вимоги до програми

Для реалізації повинні бути враховані вимоги до функціональних характеристик, умов експлуатації, складу параметрів і технічних засобів, апаратної і програмної сумісності.

4.1 Вимоги до функціональних характеристик

Додатки повинні забезпечувати наступні функції:

– відправка повідомлень іншим користувачам;

– пошук користувачів у базі динх;
– авторизація;
– реєстрація;
– зміна паролю;
– редагування даних облікового запису;

4.2 Вимоги до надійності

– забезпечувати безпеку передачі даних по мережі.
– надійність передачі даних;
– обмеження несанкціонованого доступу до даних;
– обробляти помилкові дії користувача і повідомляти про це;
– можливість самовідновлення після збоїв;
– виключати аварійні ситуації, які прямо або побічно можуть привести до псування апаратної, програмної або інформаційної складової оточення користувача;

4.3 Умови експлауатації

Умови експлуатації повинні відповідати санітарним і технічним нормам експлуатації ЕОМ. Основні вимоги для інсталяції додатку на ПК відповідають мінімальним вимогам до експлаутації операційної системи, що використовується на пристрої. Мобільний додаток працюватиме на будь-якому смартфоні із операційною системою Android або IOS.

4.4 Вимоги до інформаційної та програмної сумісності

Засіб реалізації – C#, об’єктно-орієнтована мова програмування з безпеченою системою типізації для платформи .NET.

WPF (Windows Presentation Foundation) – графічна підсистема (аналог WinForms), яка має прямі відношення до XAML. WPF включає нове ядро для заміни GDI і GDI+, використовувані в Windows Forms. WPF є високорівневим об’єктно-орієнтованим функціональним шаром, що дозволяє створювати двовимірні та тривимірні інтерфейси.

Xamarin.Forms – це платформа, призначена для створення міжплатформенних додатків для Android та iOS.

4.5 Спеціальні вимоги

Програма повинна мати дружній інтерфейс, розрахований на користувача середньої кваліфікації.

5 Вимоги до програмної документації

У момент здачі проекту замовнику надається наступний набір документів:

– методика випробувань;
– програмна система;
– керівництво користувача;
– посібник для користувача;
– опис функціоналу програми;
– текс програми.

6 Стадії та етапи розробки

Розробка програмного продукту проходить декілька стадій і етапів, які представлені в таблиці Г.1.

Таблиця Г.1 – Етапи розробки програмного продукту

Стадія розробки	Етапи робіт	Зміст робіт
1	2	3
Технічне завдання	Обґрунтування необхідності розробки програми.	Опис предметної області; призначення розробки; тестування; вимоги до програмного комплексу; стадії і етапи розробки; порядок і контроль приймання. Розробка і затвердження технічного завдання
Ескізний проект	Розробка ескізного проекту	Попередня розробка вхідних даних; уточнення середовища програмування; розробка і опис загальної алгоритмічної структури програми;
Технічний проект	Розробка технічного проекту	Уточнення структури вхідних і вихідних даних, визначення форми їх уявлення. Реалізація додатку; відладка; розробка методики випробувань.

Кінець таблиці Г.1

1	2	3
Робочий проект	Розробка програмного забезпечення	Реалізація програмного забезпечення; відладка; проведення попереднього тестування
Розробка програмної документації	Розробка документації до програмного забезпечення	Розробка необхідної документації, передбаченої технічним завданням
Тестування системи	Проведення тестування програмного забезпечення	Розробка методики тестування; проведення основних тестів; коректування програмного забезпечення
Впровадження	Підготовка і передача програми	Підготовка і передача програмного забезпечення; навчання персоналу використовуванню програмного забезпечення; внесення коректувань в програмне забезпечення і документацію

7 Порядок контролю та приймання

Контроль якості програмної системи здійснюється проведенням тестування, щоб запобігти введеню некоректних даних, перевірити надійність системи і її компонентів, тестування на повноту обміну інформацією між різними додатками.

Прийом додатку здійснюється після завантаження інсталятора на сервер і налаштування під конкретних користувачів і коротного курсу по навчанню користувачів.

Контроль здійснюється кінцевим користувачем системи, який приєднався на етапі тестування програмної системи.

ДОДАТОК Д
(обов’язковий)

ВИГЛЯД КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ

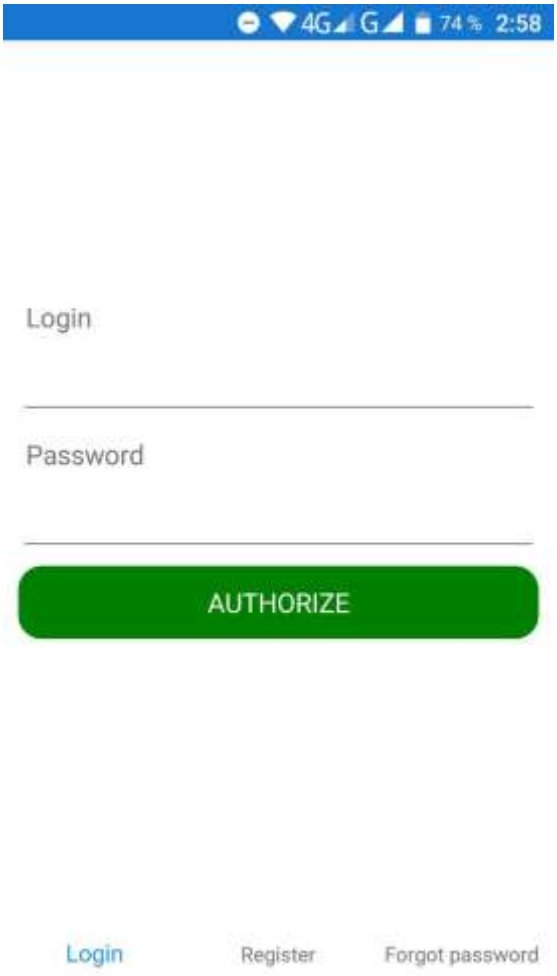


Рисунок Д.1 – Інтрфейс авторизації

Рисунок Д.2 – Інтерфейс реєстрації

Рисунок Д.3 – Інтерфейс відновлення паролю

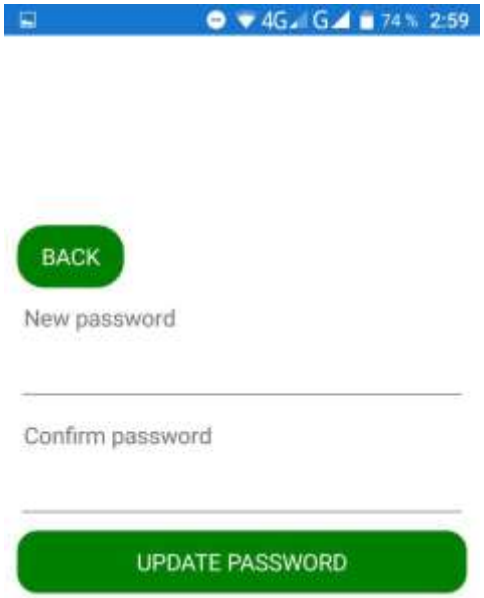


Рисунок Д.4 – Інтерфейс введення паролю

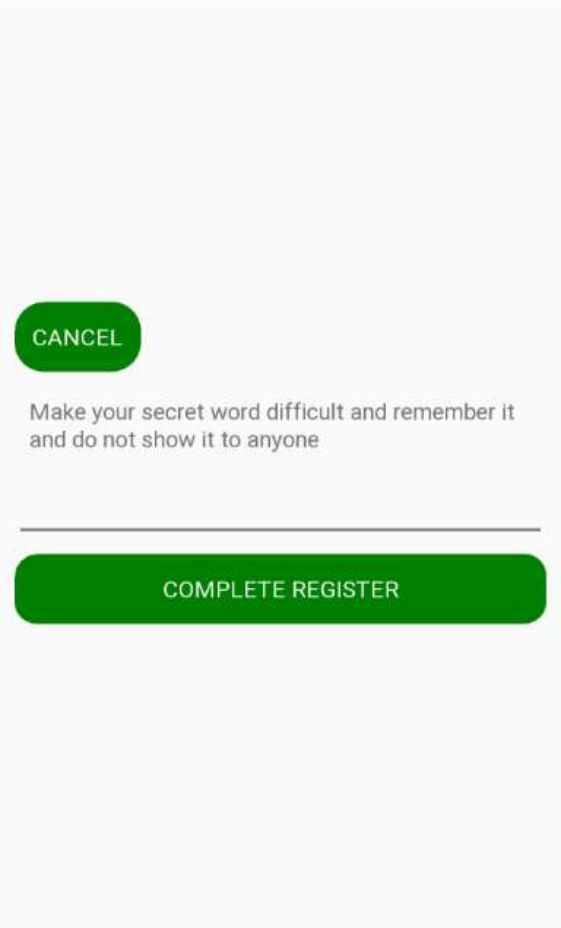


Рисунок Д.5 – Інтерфейс завершення реєстрації

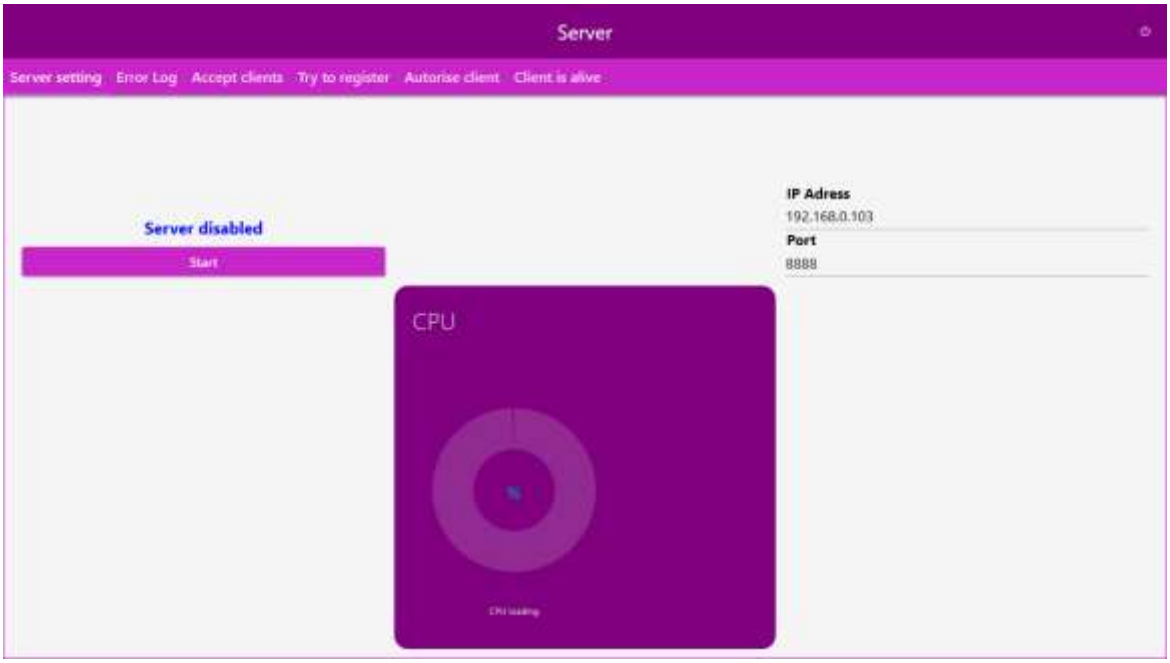


Рисунок Д.6 – Інтерфейс головного вікна

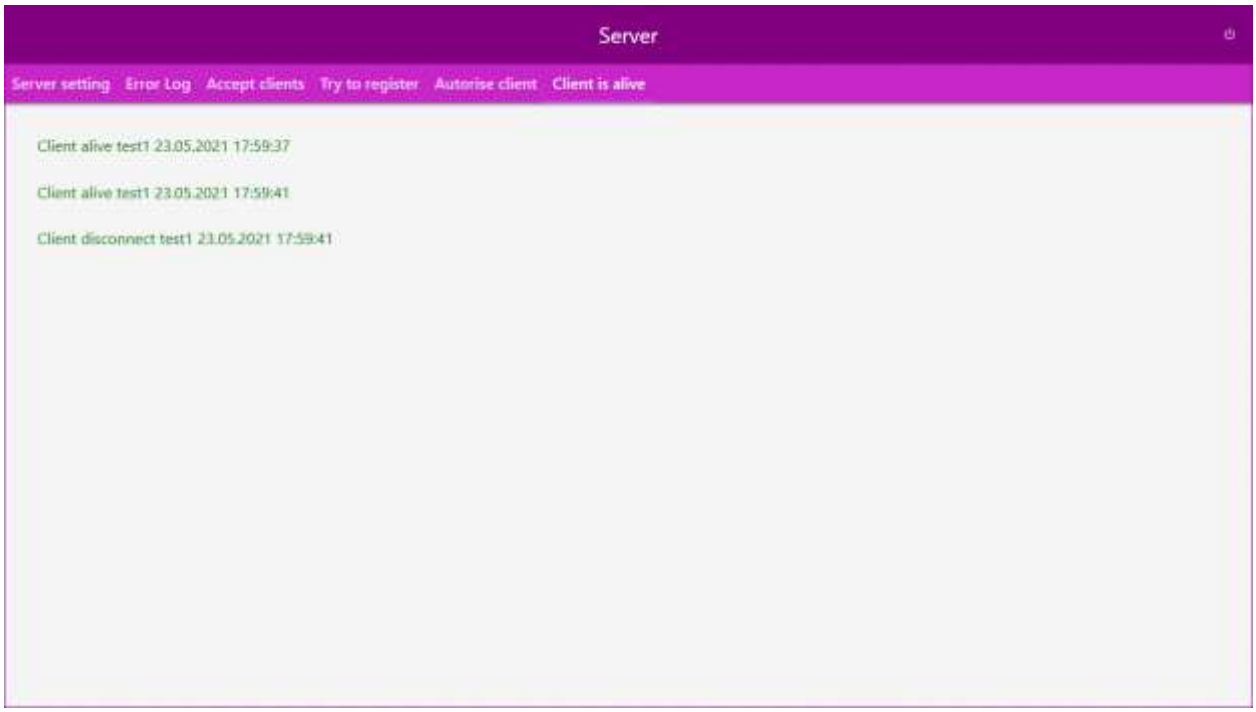


Рисунок Д.7 – Інтерфейс сповіщення

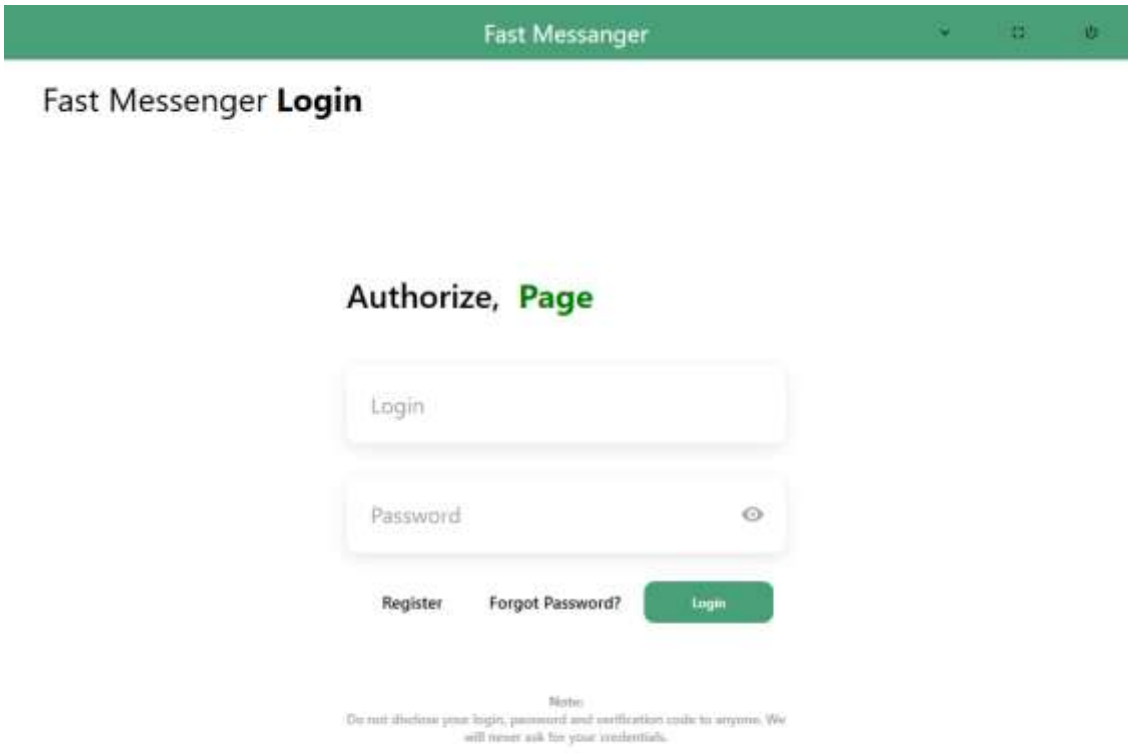


Рисунок Д.7 – Інтерфейс авторизації

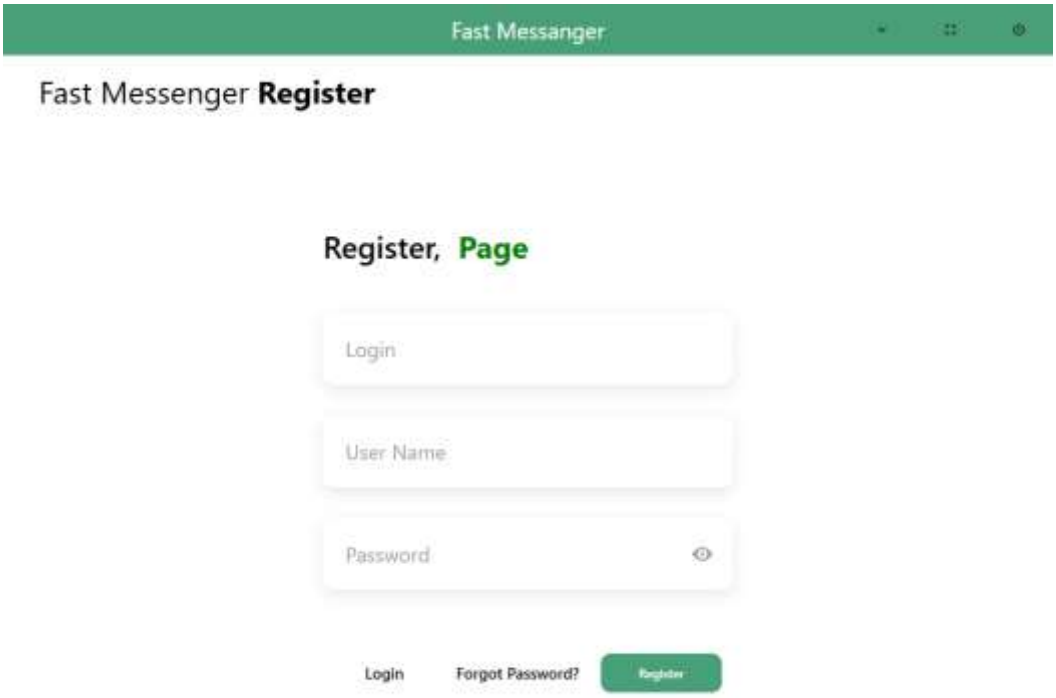


Рисунок Д.8 – Інтерфейс реєстрації

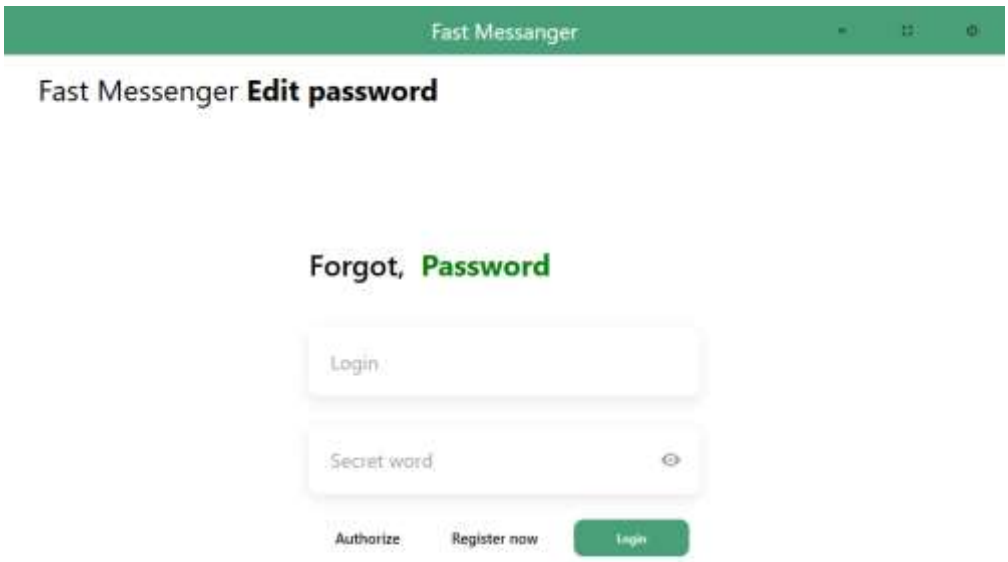


Рисунок Д.9 – Інтерфейс відновлення паролю

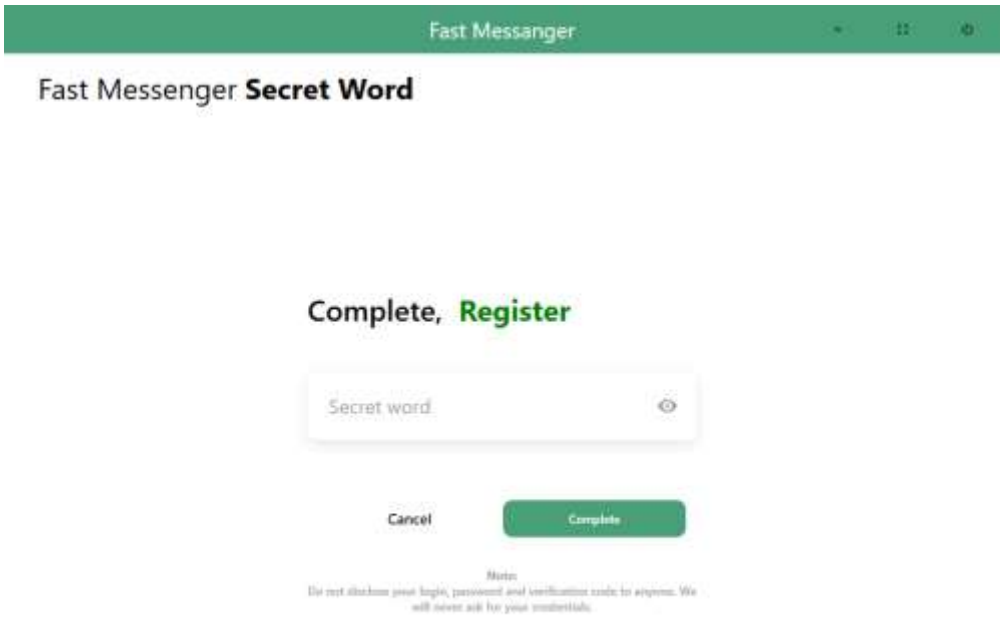


Рисунок Д.10 – Інтерфейс завершення реєстрації

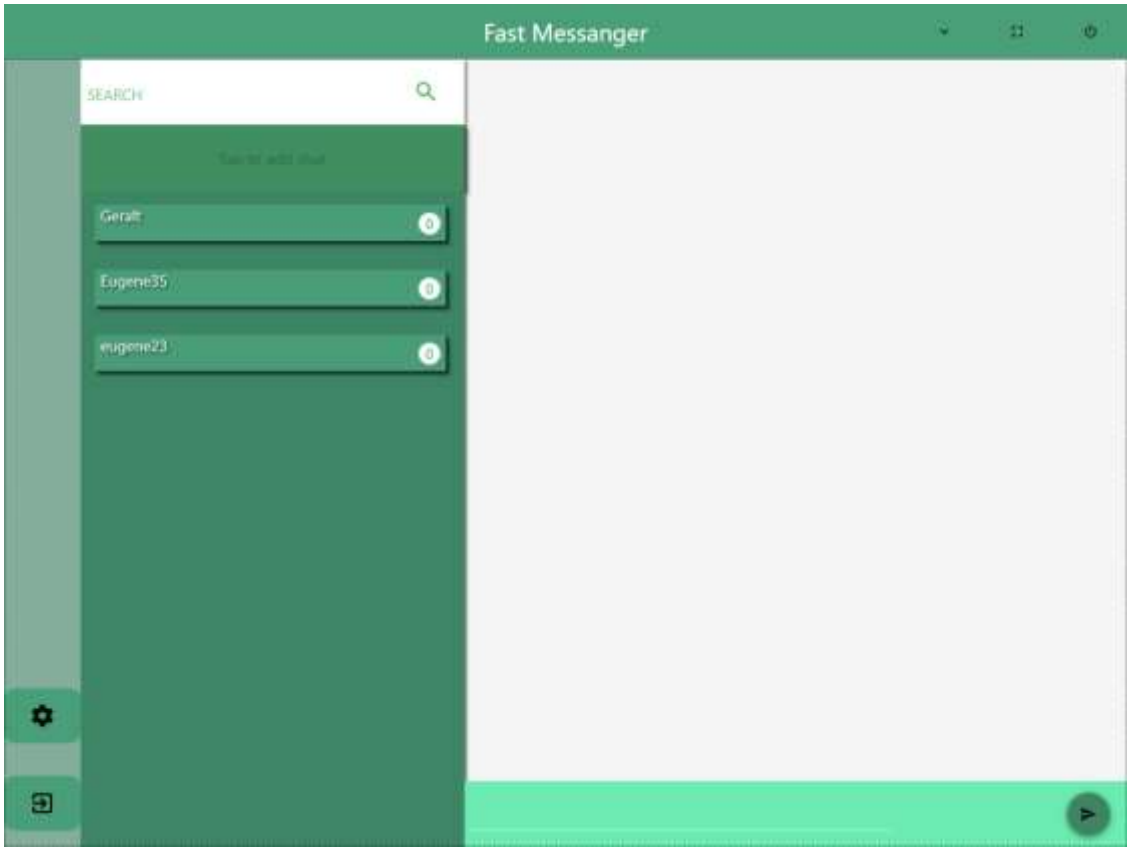


Рисунок Д.11 – Інтерфейс головного вікна

ДОДАТОК Е

(обов’язковий)

РЕЗУЛЬТАТИ МОДУЛЬНОГО ТЕСТУВАННЯ

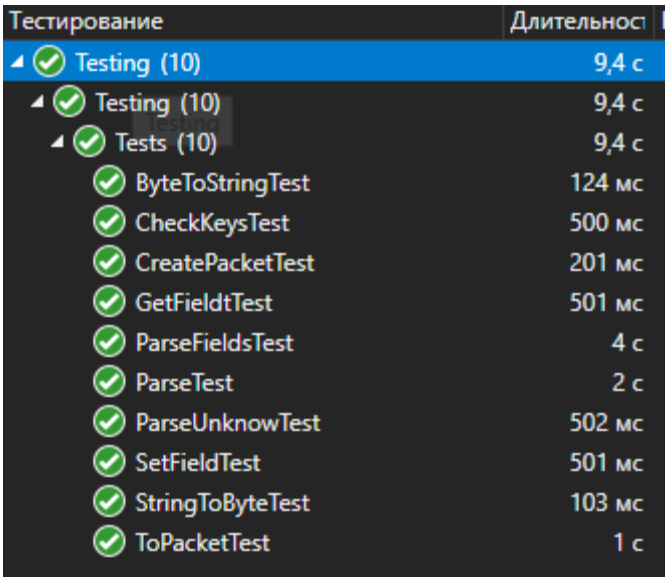


Рисунок Е.1 – Результати модульного тестування

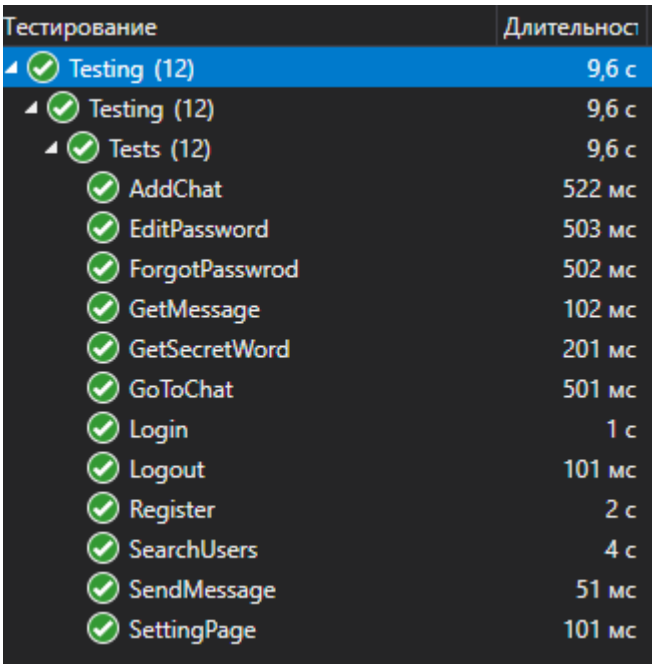


Рисунок Е.2 – Результати модульного тестування

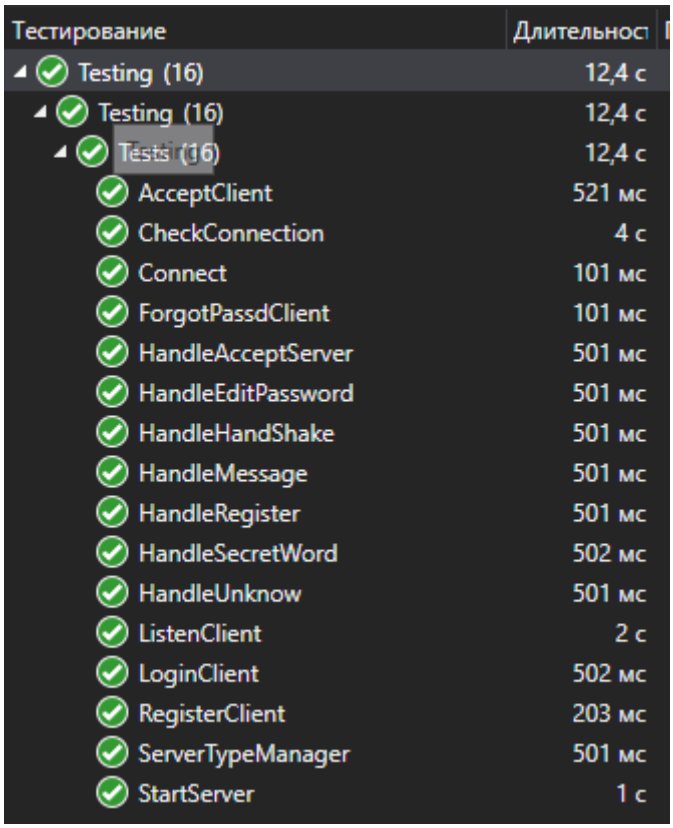


Рисунок Е.3 – Результати модульного тестування

ДОДАТОК И

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

2

Вступ

- Зараз спілкування між людьми досягло найвищого технологічного рівня. Прогрес зробив обмін повідомленнями можливим за лічені секунди. Спілкуватися в режимі реального часу люди можуть на великій відстані один від одного. Для спілкування, таким чином, були розроблені спеціальні клієнтські додатки.
- Сучасні програмні системи не завжди надають безпеку і конфіденційність при спілкувані, що змушує людей відмовлятися від них.
- Деякі програми можна встановити, як на телефон, так і на комп'ютер. Однак, кількість користувачів смартфонів збільшується і найбільший попит є на мобільні додатки.

3

Актуальність дипломного проекту

- Актуальність теми дипломного проекту полягає у розробці програмної системи, яка створить умови для надійної передачі даних. Розробка простої у використані програмної системи, яка буде доступа для користувачів.
- Мета проекту – розробити програмне забезпечення обміну повідомленнями з використанням стійких алгоритмів шифрування, яке буде включати серверний та клієнтські додатки і матиме простий та зрозумілий інтерфейс.

4

Завдання на дипломне проектування

Проблеми, які потрібно вирішити:

- провести аналіз програмних систем обміну повідомленнями;
- проаналізувати алгоритми шифрування і інтегрувати їх в свою систему;
- створити свій прикладний протокол передачі даних;
- розробити клієнтські додатки для персональних комп'ютерів та мобільних пристроїв;
- розробити серверний додаток;
- провести тестові випробування програмного забезпечення.

5

Наявне програмне забезпечення

- Telegram

Telegram - Крос-платформний месенджер, за допомогою якого ви можете надсилати повідомлення та мультимедійні файли у багатьох форматах.

Основні особливості Telegram:

- передача текстових повідомлень та голосових дзвінків;
- передача файлів різного формату;
- групові чати;
- секретні чати;
- голосові повідомлення;
- створення каналів;
- зберігання файлів на хмарному сервері;
- використання ботів.

6

Наявне програмне забезпечення

- Viber

Для авторизації користувачів та пошуку контактів програма використовує номер телефону та передає вміст телефонної книжки.

Особливості месенджера:

- підтримка систем Android, iOS та Windows Phone;
- здійснювати аудіо та відеодзвінки;
- надсилати файли та зображення;
- необмежений обмін повідомленнями;
- наявність версії для ПК;
- створення публічних каналів та чатів.

7

Наявне програмне забезпечення

- WhatsApp

WhatsApp - популярна безкоштовна система обміну повідомленнями, яка працює на мобільних пристроях та інших платформах із підтримкою голосу та відео.

Основні функції:

- спілкування за допомогою повідомлень або дзвінків;
- груповий чат;
- інтеграція контактів з адресної книги;
- пересилання контактної інформації іншим користувачам;
- адаптація дизайну інтерфейсу;
- надсилання історії листування електронною поштою;
- обмін фотографіями.

8

Аналіз способів шифрування

- Для Telegram месенджера був створений протокол MTProto, що передбачає використання декількох алгоритмів шифрування. При авторизації і аутентифікації використовуються алгоритми RSA-2048, DH-2048, а для шифрування, при передачі повідомлень через мережу використовується алгоритм AES з ключем, відомим клієнту і серверу.
- WhatsApp використовує в якості протоколу end-to-end шифрування та реалізацію алгоритму Double Ratchet від Open Whisper Systems (Signal).
- Автори Viber пішли за схожим з розробниками WhatsApp шляхом і інтегрували в свій месенджер повноцінне end-to-end шифрування, яке зберігає приватні ключі шифрування на пристроях користувачів. Розробники написали власну реалізацію алгоритму Double Ratchet.

9

Частка ринку операційних систем в світі - квітень 2021 р.

10

Висновки із аналізу існуючого програмного забезпечення

- Обов'язкові умови для роботи додатку:
  - у іншого користувача встановлена така ж програма для спілкування, переписка між користувачами різних програм неможлива;
  - інтернет – через нього і здійснюється пересилання повідомлень;
  - пристрій повинен відповідати мінімальним вимогам технічного стану.
- Основними вимогами до програмного забезпечення є:
  - необмежена відправка текстових повідомлень;
  - підтримка мобільних та комп'ютерних систем;
  - пошук інших користувачів;
  - зрозумілий та адаптивний дизайн інтерфейсу;
  - безпека передачі даних.

11

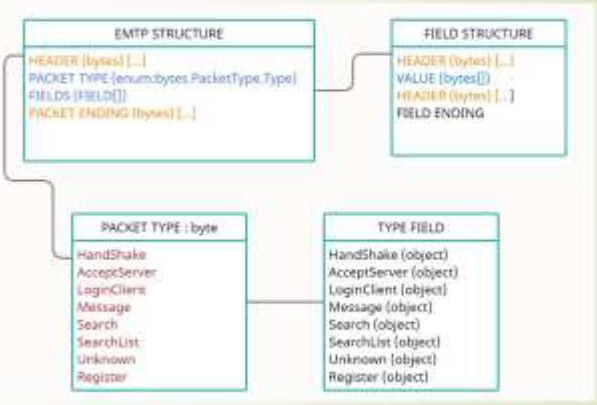
Висновки із аналізу існуючого програмного забезпечення

- Для шифрування повідомлень, потрібно використовувати протокол передачі даних, декілька стійких алгоритмів шифрування, які використовуватимуться для різних запитів до серверу. Щоб передавати дані користувача на сервер, використовуватиметься алгоритм CAST-128, а для передачі повідомлень буде використовуватись алгоритм AES.
- Беручи статистику із сайту gs.statcounter.com, робимо висновок, що найбільшу долю ринку займають пристрої із операційною системою Android – 41%, Windows – 32% та IOS – 16%. На ці операційні системи і будуть розроблені додатки.

12

Проектування

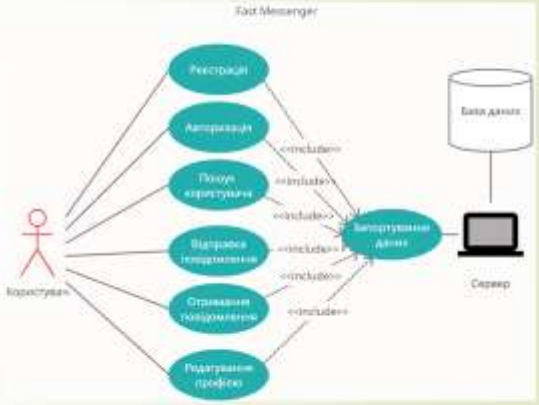
Структура пакету



13

Проектування

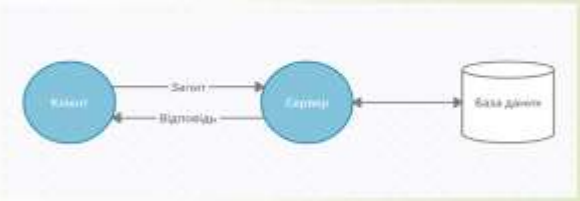
Спроектована наступна діаграма використання



14

Проектування

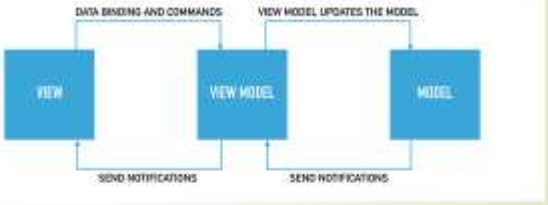
Для реалізації програмної системи вибрана клієнт-серверна архітектура



15

Проектування

Для реалізації додатків використовується шаблон проектування MVVM



16

Способи реалізації

Щоб реалізувати програмну системи, буде використана мова програмування C#, на ній буде розроблений протокол передачі даних, додатки для мобільного та настільного пристрою.

Для розробки мобільного додатку, буде використана Xamarin - це платформа з відкритим вихідним кодом для створення сучасних і продуктивних додатків для iOS, Android і Windows з .NET.

Для розробки настільного додатку, буде використана WPF - система для побудови клієнтських додатків Windows з візуально привабливими можливостями взаємодії з користувачем, графічна підсистема у складі .NET Framework, яка використовує мову XAML.

17

Висновки

В процесі виконання дипломного проектування за темою «Програмна система обміну повідомленнями з використанням стійких алгоритмів шифрування» було проведено аналіз предметної області та визначені функціональні та експлуатаційні вимоги.

Проведено аналіз існуючого програмного забезпечення, визначені переваги та способи реалізації. В результаті аналізу виведені вимоги до ПЗ. Встановлено яким чином потрібно реалізувати криптографію протоколу, що надасть безпеку спілкування. Визначені операційні системи на які потрібно розробити додатки.

Проведено проектування програмної системи, де визначені шаблони проектування, архітектура, описана діаграма використання та структура протоколу. Після проектування, описані способи реалізації програмної системи.

В результаті дипломного проектування, реалізовано програмну систему, яка дозволяє виконувати обмін повідомленнями між користувачами по мережі та забезпечую безпеку відправки даних.


18
Дякую за увагу!

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Майор Є. В.
Прізвище, ініціали

факультет ПКТС, 4 курс, група ІПЗ-17-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

04.06.2021 р.
дата

підпис

Fri Jun 04 20:49:07 EEST 2021, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

ID: 92362 Назва: Програмна система обміну повідомленнями з використанням стійких алгоритмів шифрування Додано в БД: 2021-06-04 Автора: Є. В. Майор Керівники: І. В. Гурман Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	122887	1922	5864 (5%)	91 (5%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1008188302

Дата перевірки:
05.06.2021 10:50:58 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
05.06.2021 11:03:10 EEST

ID користувача:
100005589

Назва документа: **Диплом Майор (6)**

Кількість сторінок: **142** Кількість слів: **22291** Кількість символів: **181205** Розмір файлу: **2.17 MB** ID файлу: **1008265192**

6.08%
Схожість

Найбільша схожість: **2.62%** з джерелом з Бібліотеки (ID файлу: **1008265198**)

3.17% Джерела з Інтернету	419	Сторінка 144
3.59% Джерела з Бібліотеки	64	Сторінка 147

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ
освітнього ступеня «Бакалавр»

Дипломник Майор Євген Віталійович

Тема Програмна система обміну повідомленнями з використанням стійких алгоритмів шифрування

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломного проекту:

Кількість листів креслень _____; кількість сторінок записки 144

1. Короткий зміст пояснювальної записки та прийнятих рішень У дипломній роботі проведено аналіз предметної області, описані та проаналізовані наявні програмні системи, визначені вимоги до системи. Проаналізовані архітектурні рішення для розробки, на основі яких, вибрано спосіб побудови програмного забезпечення та описані методи реалізації. Проведений детальний опис реалізації проекту, із початковим проектуванням системи. Описані методи тестування додатків і їх представлення.

2. Висновок про відповідність проекту поставленому завданню Дипломна робота освітнього ступеня «бакалавр» у повній мірі відповідає поставленому завданню, як у теоретичній, так і в практичній її частині.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі описується актуальність дипломної роботи. У першому розділі аналізується предметна область, досліджуються проблеми та способи їх вирішення, описуються вимоги до програмної системи. У другому розділі проводиться аналіз архітектурних підходів до розробки програмного забезпечення, обґрунтовується вибір шаблону проектування, проводиться декомпозиція проекту, де виконується проектування бази даних і опис інтерфейсів користувача, проводиться аналіз способів реалізації та вибір технологій для розробки. Третій розділ присвячений реалізації програмної системи, де проводиться проектування модулів, після чого детально описується реалізація цих модулів. Останній розділ висвітлює процес тестування програмного забезпечення і опис результатів роботи модулів.

4. Позитивні сторони проекту Розроблена програмна система має простий і інтуїтивний дизайн. Щоб проводити обмін повідомленнями по мережі, був розроблений власний протокол передачі даних. Використані декілька стійких алгоритмів шифрування, які використовують для різних випадків роботи програми. Варто відмітити швидкість роботи додатків та правильність вибору підходів до розробки. В дипломному проекті розроблено відразу три додатки.

5. Негативні сторони Розроблена програмна система не працюватиме на усіх популярних операційних системах. Надсилати повідомлення, можна лише користувачам із таким же додатком. Немає можливості добавити фото профілю. Користувач немає можливості фільтрувати надходженні повідомлення. Було би доцільно провести більше модульних тестів для повного розуміння роботи програми.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконане відповідно до теми дипломної роботи з дотриманням вимог стандартів. Графічне оформлення виконане на достатньому рівні. Пояснювальна записка відповідає вимогам стандартів до її оформлення.

7. Відгук про дипломний проект в цілому Дипломна робота заслуговує позитивної оцінки. Увесь матеріал дипломної роботи структурований, чіткий, послідовний та у повній мірі розкриває обрану тему, що дозволяє чітко розуміти викладений матеріал у рамках тематики дипломної роботи. Графічні матеріали дозволяють чітко побачити доцільність та ефективність рішень, які були прийняті за основу для вирішення поставленої задачі.

8. Інші зауваження

9. Оцінка дипломного проекту Розглянувши позитивні та негативні сторони представленої дипломної роботи, можна зробити висновок, що вона заслуговує на оцінку «відмінно» (4,75/A).

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) Нічепорук Андрій Олександрович, кандидат технічних наук, доцент кафедри комп'ютерної інженерії та системного програмування (КІСП) ХНУ

"02" червня 2021 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Програмна система обміну повідомленнями з використанням стійких алгоритмів шифрування»

Автор: Майор Євген Віталійович

Спеціальність: 121 – Інженерія програмного забезпечення

Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»

Науковий керівник: Гурман Іван Васильович, канд. тех. наук, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті дипломного проекту системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень в бланках (титульного аркушу, бланку завдання, в структурі підрозділів, ВСТУПУ);
2) в якості запозичень системою було зафіксовано стандартні конструкції та послідовності коду, які є базовими при реалізації настільного і мобільного додатків на вибраних технологіях, що не може розглядатися, як об'єкт авторських прав і, відповідно, їх порушення;
3) усі запозичення фрагментарні, або мають належним чином оформленні посилання.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності / схожості, складає 6,08% і адресується до 489 першоджерел, що з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь дипломного проекту.

Керівник _____ І.В. Гурман

Гарант ОП _____ Л.П. Бедратюк

Завідувач кафедри _____ Л.П. Бедратюк