МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ОВРАМЕЦЬ ІЛЛЯ ВОЛОДИМИРОВИЧ

Допускається до захисту:
завідувач кафедри
інформаційних технологій,
к. т. н., доцент
Зелінська О. В.
«_____» _____ 2024р.

РОЗРОБКА КРОСПЛАТФОРМНОГО МЕСЕНДЖЕРА НА ОСНОВІ БАГАТОШАРОВОЇ АРХІТЕКТУРИ З БАГАТОРІВНЕВИМ ШИФРУВАННЯМ ПОВІДОМЛЕНЬ.

Спеціальність 122 «Комп’ютерні науки»

Кваліфікаційна (бакалаврська) робота

Науковий керівник:
Антонов Ю.С., доцент кафедри
інформаційних технологій,
кан. фіз.-мат. н., доцент

Оцінка: _____/_____/_____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця – 2024

АНОТАЦІЯ

Оврамець І. В. Розробка кросплатформного месенджера на основі багатошарової архітектури з багаторівневим шифруванням повідомлень. Спеціальність 122 «Комп'ютерні науки», освітня програма «Сучасні інформаційні технології та програмування». Донецький національний університет імені Василя Стуса, Вінниця, 2024.

У кваліфікаційній (бакалаврській) роботі реалізується кросплатформний додаток для шифрованого обміну текстовими повідомленнями основною особливістю якого є комбінація різних методів шифрування що надасть високий рівень шифрування повідомлень, застосунок буде реалізований на платформі .NET з використанням платформи для кросплатформної розробки MAUI та мови розмітки XAML.

Ключові слова: багаторівневе шифрування, кросплатформний, .NET, MAUI.

56 с., 22 рис., 51 джерело.

ABSTRACT

Ovramets I. V. Development of a Cross-Platform Messenger Based on a Multi-Layer Architecture with Multi-Level Message Encryption. Specialty 122 "Computer Science", Educational Program "Modern Information Technologies and Programming". Vasyl Stus Donetsk National University, Vinnytsia, 2024.

In the qualification (bachelor) work, a cross-platform application for encrypted text messaging is implemented, the main feature of which is a combination of different encryption methods that will provide a high level of message encryption, the application will be implemented on the .NET platform using the platform for cross-platform development MAUI and the XAML markup language.

Keywords: multilevel encryption, cross-platform, .NET, MAUI

56 p., 22 pict., 51 source.

ЗМІСТ

ВСТУП

У сучасному світі, де кіберзлочинці активно використовують різноманітні методи для перехоплення та злому приватних комунікацій, розробка месенджера з надійним шифруванням та багатошаровою архітектурою стає крайньою необхідністю. Захист особистих даних стає важливим завданням для кожного користувача, оскільки втрата контролю над ними може призвести до серйозних наслідків, включаючи крадіжки персональних даних та фінансове шахрайства.

Зокрема в Україні підчас війни не рідкі випадки коли обмін інформацією може бути перехоплений та дешифрований ворожою стороною що ставить під загрозу життя людей як цивільних так і військових.

З огляду та такий спектр загроз шифрований месенджер забезпечить високий рівень конфіденційності та захищеності приватних розмов, що є критичним у сучасному цифровому середовищі.

Актуальність теми: Зростання кіберзагроз та необхідність захисту конфіденційних даних роблять месенджери з можливістю створення власних ключів шифрування вкрай важливими. Така функціональність забезпечує додатковий рівень безпеки для захисту приватності користувачів, корпоративних комунікацій та чутливої інформації під час обміну повідомленнями.

Мета дослідження: Розробити безпечний месенджер для обміну шифрованими повідомленнями із багатошаровою архітектурою та багаторівневим шифруванням.

Завдання дослідження:

- Проаналізувати підняті в роботі теми;
- Сформувати технічне завдання;
- Проаналізувати доступні технології та підходи;
- Розробити месенджер із багаторівневим шифруванням.

Об'єкт дослідження: Процес розробки месенджера.

Предмет дослідження: Шифрування повідомлень багаторівневим шифруванням в месенджері.

Апробація результатів дослідження: результати досліджень апробовані на V Всеукраїнської науково-практичній конференції «Прикладні інформаційні технології» Вінниця, ДонНУ імені Василя Стуса, 23.05.2024. та на міжнародному конкурсі студентських наукових робіт "Black Sea Science" 2024 року під назвою «Application for encoding and decoding text messages» за напрямком «Інформаційні технології, автоматизація і робототехніка».

Структура роботи: кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел.

Робота містить 56 сторінки, 22 рисунки та список літератури з 51 джерела.

РОЗДІЛ 1
ОГЛЯД

1.1 Актуальні платформи

У сучасному світі вибір платформ для розробки визначається не лише їхньою популярністю серед користувачів, але й технічними можливостями, які надають дані платформи для створення функціональних та безпечних додатків. На сьогоднішній день чотири основні операційні системи (ОС) Android, iOS, macOS та Windows є найбільш популярними серед користувачів [0].

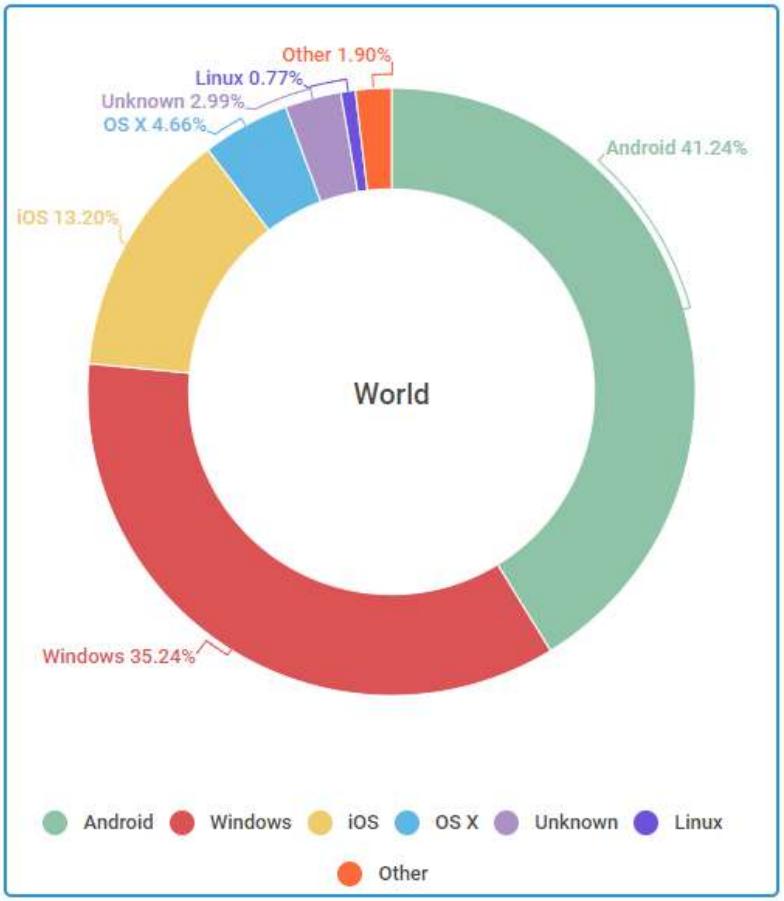

Рисунок 1.1 – Найпопулярніші у світі ОС станом на 2017 рік [1]

В Україні статистика відрізняється оскільки за даними більша частина користувачів надає перевагу платформі Windows. Проте в лідерах всеодно залишаються чотири основні платформи Android, iOS, macOS та Windows.

Розглянемо ці основні операційні системи:

Android - це операційна система, розроблена компанією Google, яка використовується в основному на мобільних пристроях, таких як смартфони та

планшети. Вона базується на ядрі Linux і має відкритий вихідний код, що дозволяє розробникам створювати різноманітні додатки та взаємодіяти з різними пристроями та сервісами. Android має величезну кількість користувачів по всьому світу і забезпечує розробникам широкі можливості для створення застосунків з різноманітними функціями [2,3].

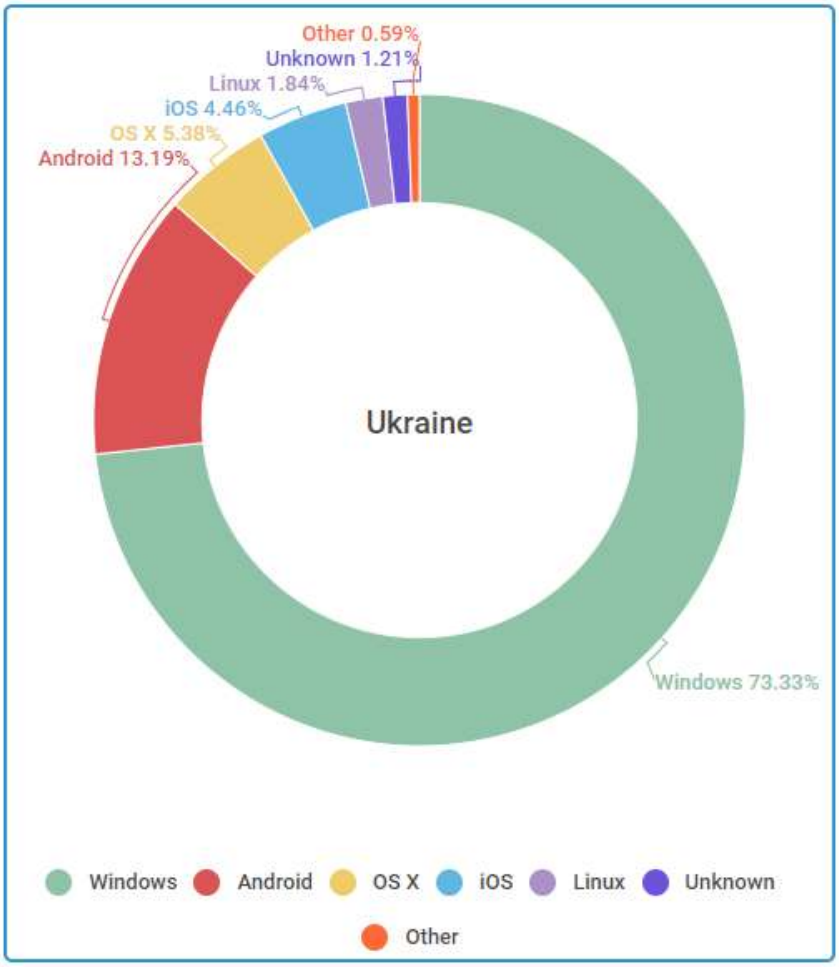

Рисунок 1.2 – Найпопулярніші в Україні ОС станом на 2017 рік [1]

iOS – це мобільна операційна система, розроблена компанією Apple, яка використовується виключно на пристроях виробництва Apple, таких як iPhone, iPad та iPod Touch. Оскільки ця платформа є закритою і контролюється Apple, вона відома своєю стабільністю, швидкістю та високим рівнем безпеки. Розробники мають доступ до інструментів розробки, таких як Xcode та Swift, які сприяють швидкому та ефективному створенню додатків [4,5].

macOS – це операційна система, розроблена компанією Apple для їхніх персональних комп'ютерів Mac. Вона відома своєю потужністю та стабільністю,

а також інтеграцією з іншими пристроями Apple, такими як iPhone та iPad. Розробники мають доступ до різноманітних інструментів для створення додатків для macOS, включаючи Xcode та мову програмування Swift. Підтримка macOS в застосунках важлива для забезпечення доступу до широкого кола користувачів, особливо серед професіоналів та креативних класів [6,7,8].

Windows – це операційна система, розроблена корпорацією Microsoft, яка використовується на багатьох персональних комп'ютерах по всьому світу. Вона відома своєю широкою розповсюдженістю та сумісністю з різноманітним апаратним забезпеченням та програмним забезпеченням. Розробники мають доступ до інструментів, таких як Visual Studio, для створення додатків для Windows, в тому числі месенджерів. Підтримка Windows дозволяє застосункам охоплювати широку аудиторію, включаючи корпоративних користувачів та геймерів [9,10].

Загалом ці чотири платформи є найбільш розповсюджених завдяки чому підходи в розробці де один код можна без переписування використовувати для створення програм під кожну із них є не тільки дуже зручним але й досить простим способом для маштабування додатків.

Також цікавість для розробників викликають менш поширені серед користувачів проте популярні серед бізнесів та різних підприємств такі операційні системи як:

Linux – це операційна система, розповсюджується під ліцензією GPL (General Public License), яка базується на ядрі Linux і використовується у багатьох дистрибутивах, таких як Ubuntu, Fedora, або Debian він використовується в наступних сферах [11].

Сервери: Linux широко використовується як операційна система для серверів, включаючи веб-сервери, бази даних, файлообмінні сервери та інші.

Робочі станції: На базі Linux розгортуються робочі станції для розробки програмного забезпечення, графічного дизайну, наукових досліджень та іншого.

Вбудовані системи: Linux використовується у вбудованих системах, таких як мобільні телефони, смартфони, маршрутизатори, телевізори та інші побутові та промислові пристрої.

FreeBSD/NetBSD – це операційні системи, що базуються на UNIX, розповсюджується під ліцензією BSD (Berkeley Software Distribution). FreeBSD і NetBSD відрізняються своїми цілями та архітектурними особливостями, але обидві пропонують високу стабільність та розширюваність вони популярні в таких сферах [12].

Сервери та мережеві пристрої: FreeBSD/NetBSD використовуються для розгортання серверів, мережевих пристроїв, маршрутизаторів, файлообмінних серверів та інших мережевих пристроїв.

Вбудовані системи: FreeBSD/NetBSD широко застосовуються у вбудованих системах, таких як маршрутизатори, пристрої зберігання даних, мережеві пристрої та інші.

QNX - це операційна система реального часу, призначена для вбудованих систем. Вона розробляється компанією BlackBerry (раніше QNX Software Systems) і відома своєю надійністю, стабільністю та використанням у важких умовах [13].

Автомобільна промисловість: QNX використовується в автомобільній промисловості для вбудованих систем управління, систем безпеки, інфотейменту та інших.

Медичне обладнання: Операційна система QNX використовується в медичному обладнанні, такому як медичні пристрої моніторингу, томографи та інші, де потрібна висока надійність та реальний час.

Інші вбудовані системи: QNX також використовується у вбудованих системах для промисловості, телекомунікаційних пристроїв, авіаційної та оборонної промисловості та інших галузях, де вимоги до надійності та реального часу є критичними.

Загалом операційні системи в основному набуває популярності в певних ринкових нішах.

1.2 Кросплатформний підхід до розробки додатків

1.2.1 Кросплатформний підхід

Кросплатформна розробка - це процес створення програмного забезпечення, здатного працювати на декількох операційних системах або платформах без необхідності розробки окремої версії для кожної з них. Цей підхід має низку переваг та недоліків, які слід враховувати [14].

Переваги кросплатформних рішень [14,15]:

1. Економія ресурсів: Розробка єдиного кодової бази для кількох платформ дозволяє значно зменшити витрати часу та матеріальних ресурсів компанії на створення окремих версій додатків.
2. Уніфікований інтерфейс користувача: Користувачі отримують однаковий інтерфейс та функціонал незалежно від платформи, на якій вони використовують додаток.
3. Спрощене оновлення та підтримка: Оскільки код є спільним для всіх платформ, оновлення та виправлення помилок можна реалізувати одночасно на всіх версіях додатку.
4. Більша доступність: Кросплатформні додатки можуть досягати ширшої аудиторії користувачів, що використовують різні операційні системи.

Недоліки крос-платформних рішень [14,15]:

1. Обмежений доступ до функціоналу: фреймворки для кросплатформної розробки не завжди забезпечують повний доступ до всіх функцій та API різних операційних систем.
2. Погіршення продуктивності: кросплатформні застосунки програють в продуктивності застосункам орієнтованим на одну платформу оскільки можуть не підтримувати найновіші оптимізовані бібліотеки для певних платформ.
3. Залежність від зовнішніх бібліотек: кросплатформні фреймворки часто покладаються на зовнішні бібліотеки та плагіни, що можуть призвести до проблем сумісності або безпеки.

4. Складність налаштування: Налаштування кросплатформних додатків може бути більш складним, оскільки потрібно враховувати особливості різних платформ.

1.2.2 Існуючі технології та фреймворки

На сьогоднішній день існує низка технологій та фреймворків для розробки кросплатформних додатків, серед яких:

React Native – це відкритий фреймворк, створений Facebook, який дозволяє розробникам створювати нативні мобільні додатки для платформ Android та iOS, використовуючи JavaScript та ReactJS. Основна ідея React Native полягає в тому, щоб дати можливість розробникам використовувати відомий їм стек технологій, такий як JavaScript та React, для створення швидких і ефективних мобільних додатків, які виглядають і працюють так само, як і нативні додатки [16,17].

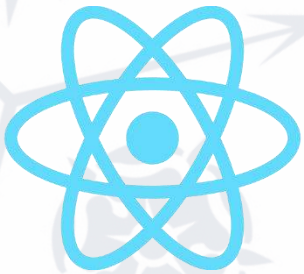

Рисунок 1.3 – Логотип React Native

Flutter – це open-source фреймворк для розробки кросплатформних мобільних додатків, створений Google. Він дозволяє створювати високопродуктивні, красиві додатки що матимуть відгук в корисувача з використанням однієї кодової бази на мові програмування Dart [18,19].

Рисунок 1.4 – Логотип Flutter

Xamarin – це потужна платформа для створення кросплатформних додатків від Microsoft. Вона дозволяє розробникам писати код одним мовою програмування (C#) та використовувати єдину кодову базу на .NET для розробки

додатків на декількох платформах: iOS, Android та Windows. Замість того, щоб писати окремий код для кожної операційної системи, розробники можуть зосередитись на створенні спільної логіки бізнес-рівня та користувацького інтерфейсу, в той час як Xamarin забезпечує відповідну "обгортку" для нативної візуалізації та взаємодії з апаратним забезпеченням на кожній цільовій платформі. [20,21].

Рисунок 1.5 – Логотип Xamarin

Apache Cordova (колишній PhoneGap) – це відкритий фреймворк із відкритим кодом для створення кросплатформних мобільних додатків. Він дозволяє розробникам використовувати веб-технології, такі як HTML, CSS та JavaScript, для розробки додатків, які потім можна упакувати як нативні додатки для декількох платформ, включаючи iOS, Android та Windows [22].

Рисунок 1.6 Логотип Apache Cordova

Ionic – це потужний open-source фреймворк для створення високопродуктивних і багатофункціональних кросплатформних мобільних додатків. Він базується на Angular та Apache Cordova, поєднуючи переваги веб-розробки та доступ до нативних можливостей мобільних пристроїв [23].

Рисунок 1.7 – Логотип Ionic

Qt – це кросплатформний фреймворк для розробки додатків на C++, що надає можливість створювати інтерфейси для різноманітних платформ, включаючи мобільні пристрої. Його потужні інструменти та бібліотеки дозволяють розробникам писати код один раз і запускати його на численних операційних системах, таких як Windows, macOS, Linux, Android та iOS, забезпечуючи при цьому нативний вигляд та поведінку додатків [25,26].

Рисунок 1.8 – Логотип Qt

На сьогоднішній день набирає така платформа як **MAUI** [27](Microsoft .NET Multi-platform App UI): MAUI є наступником Xamarin.Forms і дозволяє створювати високопродуктивні та нативні додатки для різних платформ, включаючи мобільні пристрої (Android, iOS), десктопи (Windows, macOS) та веб-браузери, використовуючи єдину кодову базу на .NET [28].

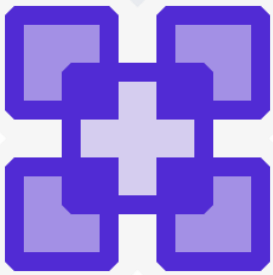

Рисунок 1.9 – Логотип MAUI

Ця технологія від Microsoft надає такі переваги [29,30]:

1. Єдина кодова база: Додатки розробляються з використанням C# та .NET, що дозволяє розробникам використовувати знайомі інструменти та навички.

2. Нативні інтерфейси користувача: MAUI забезпечує доступ до нативних елементів інтерфейсу кожної платформи, гарантуючи автентичний досвід користувача.

3. Доступ до нативних можливостей: Розробники можуть легко взаємодіяти з нативними API та функціями кожної платформи.

4. Продуктивність на рівні нативних додатків: Завдяки використанню нативних рендерерів та оптимізації, додатки на MAUI мають продуктивність, порівнянну з повністю нативними додатками.

5. Інтеграція з .NET екосистемою: MAUI тісно інтегрується з .NET та дозволяє використовувати широкий спектр бібліотек та інструментів доступні на даній платформі.

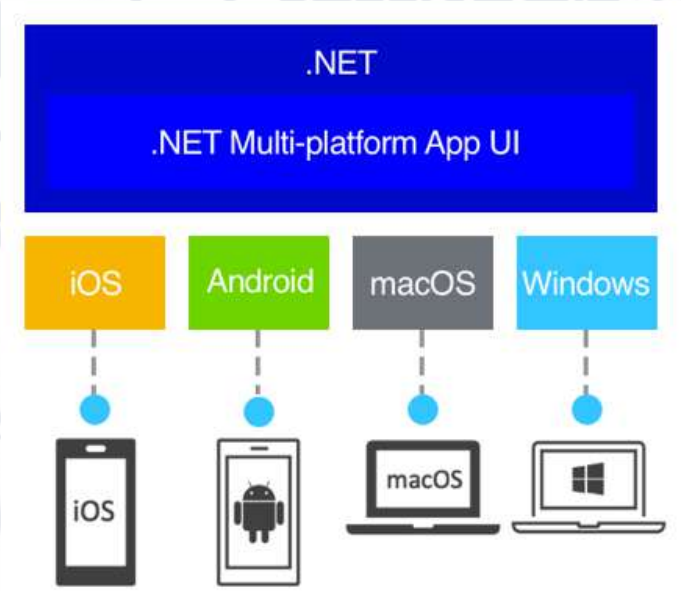

Рисунок 1.10 – Доступні в MAUI платформи [27]

MAUI є досить новою технологією, яка була випущена у травні 2022 року, але вже демонструє великий потенціал для розробки високоякісних кросплатформних додатків, особливо для команд, які вже мають досвід роботи з .NET та C# [27,29].

При розробці кросплатформного месенджера на основі багатошарової архітектури, MAUI буде одним з кращих варіантів для створення клієнтської частини, яка буде доступна на різних платформах та матиме нативний інтерфейс користувача.

1.3 Особливості розробки клієнт-серверних програм.

Розробка клієнт-серверного програмного забезпечення має ряд особливостей, які необхідно враховувати для забезпечення належної функціональності, продуктивності та безпеки системи [31].

Ключовими аспектами згідно джерела [31] є:

Архітектура – клієнт-серверна архітектура передбачає чітке розділення відповідальності між клієнтською та серверною частинами. Клієнт відповідає за представлення інтерфейсу та взаємодію з користувачем, тоді як сервер обробляє запити, керує даними та виконує основну логіку додатку.

Мережева взаємодія – клієнти та сервер повинні ефективно взаємодіяти через мережу, використовуючи протоколи, такі як HTTP, WebSocket або власні протоколи. Важливо забезпечити надійну передачу даних, керування помилками та безпеку з'єднання.

Масштабованість – клієнт-серверні системи часто повинні обслуговувати велику кількість клієнтів одночасно. Необхідно враховувати масштабованість серверної частини, використовуючи підходи, такі як горизонтальне масштабування, балансування навантаження та розподілену архітектуру.

Безпека – забезпечення безпеки є критичним аспектом клієнт-серверних додатків, особливо якщо вони обробляють конфіденційні дані чи здійснюють фінансові транзакції. Необхідно впроваджувати заходи безпеки, такі як шифрування даних, аутентифікація користувачів, запобігання вразливостям та захист від атак.

Сумісність та кросплатформність – клієнтська частина додатку може працювати на різних платформах (веб, мобільні пристрої, десктопні додатки), тому важливо забезпечити сумісність та кросплатформність, використовуючи відповідні технології та підходи до розробки.

Синхронізація даних – у багатьох клієнт-серверних додатках необхідно забезпечити синхронізацію даних між клієнтами та сервером, щоб гарантувати актуальність та узгодженість інформації.

Оптимізація продуктивності – важливо оптимізувати продуктивність як клієнтської, так і серверної частини, щоб забезпечити швидку відповідь системи та ефективне використання ресурсів.

Керування станом – у клієнт-серверних додатках необхідно ретельно керувати станом додатку, зберігаючи відповідні дані на клієнті та/або на сервері, залежно від вимог до функціональності та безпеки.

Відновлення після збоїв – клієнт-серверні додатки повинні бути стійкими до збоїв мережі, апаратних або програмних помилок, забезпечуючи механізми відновлення та зберігання критичних даних.

Керування версіями та розгортання – з огляду на складність клієнт-серверних додатків, необхідно ретельно керувати процесами оновлення, тестування та розгортання як клієнтської, так і серверної частини.

Ці особливості підкреслюють важливість ретельного планування, проектування та реалізації клієнт-серверних програм для забезпечення належної функціональності, продуктивності, безпеки та масштабованості системи.

1.4 Багатошарова архітектура

Багатошарова архітектура є потужним підходом до проектування та розробки складних програмних систем, який забезпечує модульність, гнучкість та можливість розширення. Ця архітектура ґрунтується на принципі розділення відповідальності, де компоненти системи організовані у логічні шари або рівні абстракції. Кожен шар має чітко визначені обов'язки та взаємодіє лише з суміжними шарами, дотримуючись певних правил та інтерфейсів.

Загальноприйнятою практикою є розподіл системи на три основні шари: представлення, бізнес-логіка та доступ до даних [32]. Однак, залежно від складності та специфічних вимог проекту, можуть використовуватися додаткові шари, такі як шар сервісів, шар інтеграції чи шар доменних об'єктів.

1. «Шар представлення (Presentation Layer)»: Цей шар відповідає за взаємодію з користувачем та візуалізацію даних. Він містить компоненти інтерфейсу користувача, такі як форми, меню та діалогові вікна. Цей шар не

містить жодної бізнес-логіки і виступає лише як посередник між користувачем та іншими шарами системи.

2. «Шар бізнес-логіки (Business Logic Layer)»: Тут міститься основна функціональність додатку, правила та обробка даних. Цей шар відокремлений від інших шарів і не залежить від специфіки представлення даних чи способу зберігання. Він забезпечує централізоване управління бізнес-процесами та гарантує цілісність даних.

3. «Шар доступу до даних (Data Access Layer)»: Цей шар відповідає за взаємодію з джерелами даних, такими як бази даних, файлові системи чи зовнішні сервіси. Він абстрагує деталі доступу до даних від решти системи, забезпечуючи єдиний інтерфейс для роботи з різними типами сховищ даних.

Багатошарова архітектура забезпечує низку переваг [33], серед яких:

1. «Модульність»: Кожен шар є окремим модулем, який можна розробляти, тестувати та підтримувати незалежно від інших шарів.

2. «Гнучкість»: Архітектура дозволяє легко замінювати або модифікувати окремі шари без впливу на інші частини системи.

3. «Масштабованість»: Окремі шари можна розгортати на різних серверах або ресурсах, забезпечуючи горизонтальне масштабування та оптимізацію продуктивності.

4. «Повторне використання»: Компоненти та логіка одного шару можуть бути повторно використані в інших проектах або додатках.

5. «Безпека»: Розділення відповідальності та інкапсуляція даних у окремих шарах покращують безпеку системи, обмежуючи доступ до критичних компонентів.

6. «Полегшене тестування»: Завдяки чіткому розділенню обов'язків, окремі шари можна тестувати незалежно один від одного, спрощуючи процес тестування та усунення дефектів.

Хоча багатошарова архітектура може бути складнішою в реалізації порівняно з монолітними рішеннями, вона забезпечує кращу підтримку, масштабованість та гнучкість у довгостроковій перспективі. Ретельне

проектування та дотримання принципів розподілу відповідальності є ключовими для успішної реалізації багатошарової архітектури в програмних системах.

1.5 Шифрування повідомлень

Багаторівневе шифрування є підходом, який поєднує кілька різних криптографічних методів і технологій для забезпечення підвищеної безпеки та конфіденційності повідомлень у месенджері. Цей підхід передбачає застосування декількох рівнів шифрування, кожен з яких вирішує певні завдання безпеки та має свої переваги згідно джерел [34,35].

Розглянемо декілька методів шифрування повідомлень [34].

1. **Симетричне шифрування повідомлень**

- Використання потужних алгоритмів симетричного шифрування, таких як AES-256 чи ChaCha20, для забезпечення конфіденційності вмісту повідомлень.
- Сеансові ключі симетричного шифрування генеруються випадковим чином для кожної сесії обміну повідомленнями.

2. **Асиметричне шифрування та обмін ключами**

- Використання алгоритмів асиметричного шифрування, наприклад, криптографії еліптичних кривих (ECC), для безпечного обміну сеансовими ключами симетричного шифрування між учасниками.
- Протоколи обміну ключами, такі як Диффі-Хеллман (ECDH), забезпечують захищений розподіл ключів без передачі їх через незахищені канали.

3. **Цифрові підписи та автентифікація**

- Використання цифрових підписів, заснованих на асиметричних алгоритмах (наприклад, ECDSA), для підтвердження автентичності відправника та цілісності повідомлень.
- Впровадження інфраструктури відкритих ключів (PKI) або інших механізмів розповсюдження та перевірки відкритих ключів.

4. **Хешування та перевірка цілісності**

- Застосування криптографічних геш-функцій (наприклад, SHA-256 або SHA-3) для генерування цифрових відбитків повідомлень.
- Цифрові відбитки використовуються для перевірки цілісності повідомлень і виявлення будь-яких змін або пошкоджень під час передавання.

5. **Забезпечення прямої секретності (Perfect Forward Secrecy)**

- Використання протоколів та алгоритмів, які забезпечують пряму секретність, таких як Signal Protocol або Off-the-Record Messaging (OTR).
- Пряма секретність гарантує, що навіть якщо довгострокові ключі опиняться в руках зловмисників, вони не зможуть розшифрувати попередні сеанси зв'язку.

6. **Безпечне видалення повідомлень**

- Впровадження механізмів для безпечного видалення повідомлень з пристроїв учасників та серверів після завершення сеансу зв'язку.
- Забезпечення неможливості відновлення видалених повідомлень у майбутньому.

7. **Захист метаданих та анонімність**

- Використання технологій, таких як Tor або I2P, для приховування метаданих зв'язку (IP-адреси, часові мітки тощо) від стороннього спостереження.
- Впровадження механізмів приховування взаємозв'язку між учасниками (relationship anonymity).

Багаторівневе шифрування поєднує переваги різних криптографічних методів, забезпечуючи комплексний захист конфіденційності, цілісності, автентичності та анонімності повідомлень у месенджері. Цей підхід дозволяє досягти високого рівня безпеки та стійкості до різних видів атак та загроз.

РОЗДІЛ 2
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

2.1 Постановка задачі

Необхідно розробити кросплатформний месенджер з багаторівневим шифруванням повідомлень на основі багатошарової архітектури. Цей месенджер повинен забезпечити високий рівень конфіденційності та безпеки передачі даних, гарантуючи, що повідомлення залишатимуться захищеними від несанкціонованого доступу та перехоплення.

Повідомлення будуть надсилатися по принципу Broadcast що дозволить на основі конкретного сервера створювати аналог групового чату який буде мати високий рівень захисту за рахунок того що ключі від шифрування будуть тільки на стороні клієнту.

Основною особливістю чату буде можливість поєднувати декілька методів шифрування за рахунок багаторівневої архітектури застосунку та структурних особливостей.

Потрібно реалізувати кілька методів шифрування зокрема RSA шифрування, шифр перестановки, та модифікований алгоритм Шеннона–Фано.

2.2 Архітектурні та проєктувальні патерни

Для сучасних застосунків дуже важливу роль відіграють патерни проектування оскільки вони забезпечують простоту в впроваджені нових можливостей та функціоналу в програму пришвидшують роботу над нею. Одним із таких патернів є MVVM (Model-View-ViewModel) який вперше був представлений Джоном Госманом та є модифікацією шаблону Presentation Model [36]. Розглянемо його більш детально.

MVVM (Model-View-ViewModel) є широко використовуваним патерном проектування в розробці користувацьких інтерфейсів, який забезпечує розділення відповідальностей та полегшує тестування, підтримку та

розширюваність додатків. Основна ідея MVVM полягає в тому, щоб відокремити логіку представлення (View) від логіки бізнес-моделі (Model) за допомогою проміжного шару ViewModel.

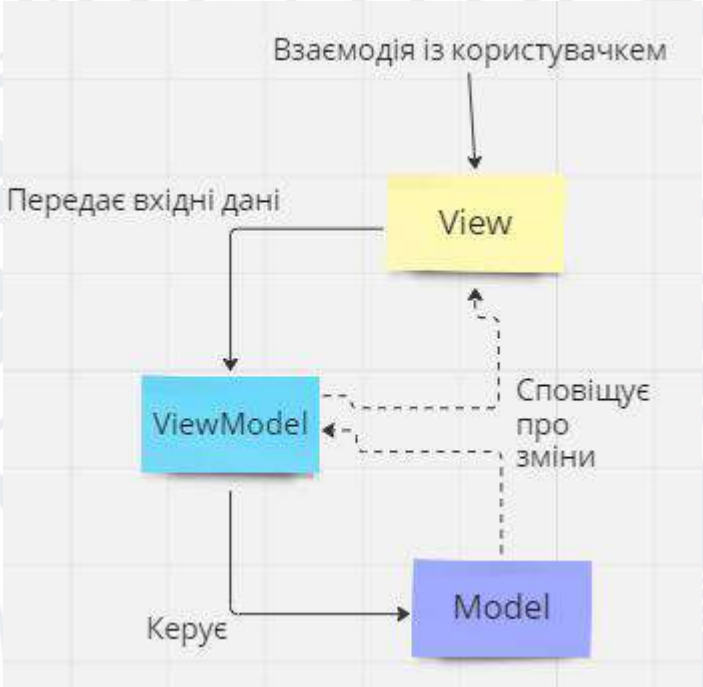

Рисунок 2.1 – Діаграма взаємодії між компонентами шаблону MVVM [36]

Патерн MVVM ґрунтується на кількох ключових принципах, які визначають роботу його компонентів та їхню взаємодію [37]:

1. **Розділення відповідальностей**:
   - Model: Відповідає за інкапсуляцію бізнес-логіки (алгоритмів шифрування) та даних додатку.
   - View: Відповідає лише за відображення користувацького інтерфейсу та елементів керування.
   - ViewModel: Є проміжним шаром між Model та View, відповідає за підготовку даних для представлення та управління станами та командами інтерфейсу.
2. **Двостороннє зв'язування даних**: View і ViewModel пов'язані за допомогою механізму двостороннього зв'язування даних. Зміни в моделі даних ViewModel автоматично відображаються у View, а зміни, зроблені у View, автоматично оновлюють відповідні властивості у ViewModel.

3. **Команди**: ViewModel інкапсулює команди, які представляють дії користувача в інтерфейсі (наприклад, натискання кнопки або вибір з меню). View пов'язується з цими командами, дозволяючи користувачеві ініціювати відповідну поведінку у ViewModel.

4. **Незалежність від платформи**: Model і ViewModel не мають жодних залежностей від конкретної платформи користувацького інтерфейсу. Їх можна повторно використовувати на різних платформах, створюючи нові представлення для кожної платформи.

5. **Життєвий цикл** ViewModel: ViewModel має власний життєвий цикл, відокремлений від View. Він може ініціалізуватися перед створенням View, підписуватися на події моделі даних та виконувати необхідні дії при завершенні своєї роботи.

6. **Одиночна точка входу**: ViewModel є єдиною точкою входу для View. Представлення взаємодіє лише з ViewModel, а не безпосередньо з Model чи іншими компонентами додатку.

7. **Тестованість**: Оскільки ViewModel не залежить від конкретного користувацького інтерфейсу, його можна легко тестувати за допомогою модульних тестів, перевіряючи логіку та поведінку без необхідності запускати повний інтерфейс.

8. **Багаторазове використання**: Завдяки розділенню відповідальностей та незалежності від платформи, компоненти MVVM (особливо Model і ViewModel) можна легко повторно використовувати в інших частинах додатку або навіть в інших проектах.

Загалом даний підхід буде одним із кращих принципів для побудови клієнтської частини месенджера оскільки дозволить розділити код на окремі шари та спростить процес розробки. Також забезпечить можливість в подальшому створені алгоритми перенести в інші версії даного месенджера та надасть доступ до кожного алгоритму для можливості перенесення їх в інші програми.

2.3 Стек технологій для розробки

Для написання коду та його відлагодження буде використовуватися мова програмування C# [38,39,40] та середовище розробки Visual Studio.

C# (C Sharp) - це об'єктно-орієнтована мова програмування, розроблена компанією Microsoft [38]. Вона була представлена в 2000 році і стала однією з основних мов для розробки програмного забезпечення для платформи .NET Core.

.NET Core - це відкрита програмна платформа, розроблена корпорацією Microsoft для створення програмного забезпечення під різними операційними системами, включаючи Windows, Linux і macOS [39]. Вона надає велику кількість бібліотек і сервісів для розробки програм, таких як графічні інтерфейси, обробка даних, мережеві операції та багато іншого.

C# є однією з основних мов програмування для платформ .NET Core. Вона є високорівневою мовою з вбудованим збирачем сміття і надає велику кількість інструментів для розробки програм [40], включаючи:

Високорівневу архітектуру, що дозволяє програмістам працювати на більш високому рівні абстракції і зосередитися на логіці програми, а не на деталях роботи з пам'яттю і обробкою помилок.

Об'єктно-орієнтований підхід, що дозволяє організувати програмний код у вигляді об'єктів, що взаємодіють між собою, і спрощує розробку та підтримку застосунку.

Розширені можливості розробки під різні платформи та великий обсяг користувацьких розширень, які можна підвантажити за допомогою вільної системи керування пакетами NuGet.

Visual Studio – це інтегроване середовище розробки (Integrated Development Environment, IDE), розроблене компанією Microsoft [41]. Це програмне забезпечення призначене для розробки програмного забезпечення для різних платформ, таких як Windows, Android, iOS, веб-програмування та інших. Visual Studio надає широкий набір інструментів для написання коду, відлагодження, тестування та розгортання програм.

Для створення кросплатформного застосунку буде використано кросплатформна платформа MAUI та мова розмітки XAML що в ній використовується.

.NET MAUI (Multi-platform App UI) є новим фреймворком, який доступний з .NET 6 [28]. Це високорівневий набір інструментів для створення мобільних, настільних та вбудованих кросплатформних застосунків. Він використовує мову програмування C# для написання бізнес-логіки та мову розмітки XAML для написання інтерфейсу користувача.

Мова розмітки XAML (Extensible Application Markup Language) широко використовується в розробці застосунків для опису інтерфейсу користувача. Вона дозволяє визначати структуру та вигляд елементів інтерфейсу, а також взаємозв'язки між ними [42,43]. Під час розробки застосунків на .NET MAUI, використання XAML спрощує процес розробки і забезпечує більшу чіткість коду.

Основними елементами розмітки в XAML згідно джерела [42] є:

Елементи контейнерів: Ці елементи дозволяють організувати та групувати інші елементи. Деякі з найпоширеніших контейнерів включають Grid, StackPanel, WrapPanel, Canvas тощо.

Елементи управління: Ці елементи представляють різноманітні елементи інтерфейсу, такі як кнопки, тексти, списки, текстові поля тощо. Найпоширеніші елементи управління включають Button, TextBlock, TextBox, ComboBox, ListView тощо.

Елементи малювання і графіки: XAML дозволяє створювати елементи малювання та графіки, такі як лінії, криві, фігури тощо. Деякі з цих елементів включають Line, Ellipse, Rectangle, Path тощо.

Елементи макетування: Ці елементи дозволяють контролювати розміщення та розміри інших елементів на макеті. Наприклад, Margin визначає відступи навколо елемента, HorizontalAlignment та VerticalAlignment визначають вирівнювання елемента в макеті.

Ресурси: Використання ресурсів дозволяє визначати стилі, шаблони, кольори та інші властивості, які можна повторно використовувати у всьому додатку.

Події і прив'язки даних: XAML дозволяє встановлювати обробники подій та прив'язки даних, які дозволяють зв'язувати дані з відповідними елементами інтерфейсу.

Програмне забезпечення для контролю версій GIT.

GIT (система контролю версій) – це розподілена система керування версіями, що використовується для відстеження змін у вихідному коді та координації роботи кількох розробників над проєктом. Вона дозволяє зберігати історію змін файлів, відстежувати, хто і коли вносив зміни, а також об'єднувати зміни, внесені різними розробниками [44,45].

Основні поняття в GIT включають у себе репозиторій (repository), який представляє собою сховище файлів та історії змін, коміт (commit) - це збереження змін в репозиторії, гілки (branch) - це паралельні лінії розробки, які дозволяють працювати над різними версіями проєкту незалежно, та злиття (merge), яке дозволяє об'єднати зміни з різних гілок.

GIT також надає багато інструментів для керування проєктами, таких як можливість повернутися до попередніх версій, створення та застосування патчів, а також віддалена співпраця з іншими розробниками через віддалені репозиторії.

Ключовою особливістю GIT є його розподілена природа, яка дозволяє кожному розробнику мати повну копію репозиторія, що забезпечує відносну незалежність та стійкість до збоїв мережі.

Спосіб з’єднання із сервером буде реалізовано за допомогою TSP протоколу.

TCP (Transmission Control Protocol) – це один із ключових протоколів комп'ютерних мереж. Він забезпечує надійний, послідовний та точний обмін даними між різними комп'ютерами у мережі Інтернет або будь-якій іншій комп'ютерній мережі [46] .

Основні особливості TCP згідно [47]:

Надійність: TCP гарантує доставку даних у порядку їх відправлення та впорядкованості. Це досягається за допомогою механізмів перевірки доставки та повторної відправки в разі втрати пакетів даних.

Контроль потоку: TCP автоматично контролює швидкість передачі даних між відправником та отримувачем, щоб уникнути перевантаження мережі або втрати даних.

Встановлення та розрив з'єднання: Перед передачею даних TCP спочатку встановлює з'єднання між відправником і отримувачем, а після завершення передачі даних відправник і отримувач розривають це з'єднання.

Підтримка даних в потоці (stream): TCP розглядає дані як потік байтів, тобто послідовний набір даних без визначених меж.

Розподіленим управлінням портами (port): Кожен процес, що використовує TCP для комунікації, ідентифікується за допомогою номера порту. Це дозволяє одному комп'ютеру взаємодіяти з кількома процесами на іншому комп'ютері.

2.4 Огляд методів шифрування

Алгоритм шифрування RSA.

Алгоритм RSA (Rivest-Shamir-Adleman) – один із найбільш відомих та найбільш використовуваних асиметричних криптографічних алгоритмів. Він був розроблений у 1977 році Рональдом Райвестом, Аді Шаміром та Леонардом Адлеманом [48].

Історія створення алгоритму RSA почалася у 1970-х роках, коли Рональд Райвест, який на той момент був аспірантом у Массачусетському технологічному інституті (MIT), зацікавився проблемами криптографії та безпеки інформації. У 1973 році він вже розробив алгоритм шифрування під назвою "Шифр RSA" у співпраці з Кліфордом Коксом.

Але остаточна версія алгоритму, яка отримала назву RSA, була розроблена у 1977 році, коли Аді Шамір та Леонард Адлеман, які тоді працювали в університеті Массачусетса, долучилися до роботи над проектом. Вони побачили

великий потенціал у роботі Райвеста та допомогли вдосконалити та випробувати алгоритм. Назва RSA складається з перших літер їхніх прізвищ.

Після публікації їхньої роботи у журналі Communications of the ACM у 1978 році, алгоритм RSA став відомим широкій громадськості та швидко здобув популярність як ефективний метод шифрування та цифрового підпису. Він залишається одним з найбільш важливих алгоритмів в сучасній криптографії і використовується в багатьох сферах, включаючи інтернет-протоколи, електронну комерцію та захист конфіденційної інформації. Алгоритм кодування RSA складається з кількох кроків:

1. Вибирається два великі прості числа, наприклад p і q.
2. Обчислюється добуток n = p * q.
3. Обчислюється функція Ейлера φ(n) = (p - 1) * (q - 1).
4. Вибирається ціле число e, таке, що $1 < e < \varphi(n)$ і найбільший спільний дільник $(e, \varphi(n)) = 1$. Число e називається відкритим ключем.
5. Обчислюється число d, таке, що $d * e \equiv 1 \pmod{\varphi(n)}$. Число d називається закритим ключем.

Таким чином, відкритий ключ складається з пари чисел (e, n), а закритий ключ - з пари (d, n).

Кодування повідомлення m виконується шляхом обчислення c = m^e (mod n). Це операція може бути виконана будь-ким, хто знає відкритий ключ (e, n).

Декодування зашифрованого повідомлення c виконується шляхом обчислення m = c^d (mod n). Ця операція може бути виконана тільки тим, хто знає закритий ключ (d, n).

Безпека RSA базується на складності факторизації великих чисел для визначення p і q із добутку n. Якщо факторизувати n неможливо, знайти d із e та n обчислювально складно.

Перестановочний шифр.

Перестановочний шифр - це метод шифрування, де символи повідомлення переставляються в певному порядку відповідно до ключа шифрування. Кожному символу відповідає нове місце у повідомленні згідно з правилами,

встановленими ключем. Наприклад, у шифрі Цезаря, кожна буква алфавіту зсувається на певне число позицій у напрямку абетки [49].

У більш загальному сенсі, перестановочні шифри можуть використовувати різні методи перестановки, такі як перемішування символів у випадковому порядку або використання складніших правил перестановки залежно від ключа. Перестановочні шифри можуть бути як простими, так і складними залежно від методу перестановки та складності ключа.

Алгоритм Шеннона–Фано

Алгоритм Шеннона–Фано – популярний метод стиснення даних, розроблений Клодом Шенноном і Робертом Фано в 1940-х роках. Алгоритм призначений для стиснення даних шляхом призначення коротших кодів символам, які частіше зустрічаються в повідомленні, і довших кодів символам, які зустрічаються рідше.

<table>
<tr><th>a_i</th><th>$p(a_i)$</th><th>1</th><th>2</th><th>3</th><th>4</th><th>Code</th></tr>
<tr><td>a_1</td><td>0.36</td><td rowspan="2">0</td><td colspan="3">00</td><td>00</td></tr>
<tr><td>a_2</td><td>0.18</td><td colspan="3">01</td><td>01</td></tr>
<tr><td>a_3</td><td>0.18</td><td rowspan="6">1</td><td colspan="3">10</td><td>10</td></tr>
<tr><td>a_4</td><td>0.12</td><td rowspan="5">11</td><td colspan="2">110</td><td>110</td></tr>
<tr><td>a_5</td><td>0.09</td><td rowspan="2">111</td><td>1110</td><td>1110</td></tr>
<tr><td>a_6</td><td>0.07</td><td>1111</td><td>1111</td></tr>
</table>

Рисунок 2.2 – Приклад кодування 6 символів.

Пояснення до рисунку:

a_i = Символ який потрібно закодувати.

$p(a_i)$ = Ймовірність появи символу в тексті.

Алгоритм працює на основі двох ключових принципів: ймовірність та ентропія.

Ймовірність – алгоритм починається з аналізу повідомлення, яке потрібно стиснути, і визначення ймовірності появи для кожного символу в повідомленні. Він призначає коди кожному символу на основі їхньої ймовірності, причому

символам, які зустрічаються частіше, призначаються коротші коди, а тим, що зустрічаються рідше, призначаються довші коди.

Ентропія – це міра випадковості або невизначеності в повідомленні. У контексті алгоритму Шеннона-Фано ентропія використовується для призначення кодів, які враховують ймовірність кожного символу в повідомленні. Символам з більшою ймовірністю призначатимуться коротші коди, а символам із меншою ймовірністю – довші коди.

Призначаючи коротші коди символам із вищою ймовірністю та довші коди символам із меншою ймовірністю, алгоритм Шеннона-Фано гарантує, що стиснене повідомлення використовує загальну кількість бітів. Це призводить до більш ефективного представлення вихідного повідомлення, зменшення вимог до пам’яті та підвищення швидкості передачі даних.

Загалом, принципи ймовірності та ентропії є важливими для роботи алгоритму Шеннона–Фано. Аналізуючи ймовірність символів у повідомленні та використовуючи ентропію для призначення кодів на основі їх ймовірності, алгоритм здатний ефективно стискати дані, зберігаючи при цьому високий рівень точності вихідного повідомлення.

На додаток до принципів ймовірності та ентропії, алгоритм Шеннона-Фано також використовує ієрархічну схему кодування для представлення символів у повідомленні. Схема кодування призначає коротші коди символам, які частіше зустрічаються, і довші коди символам, які зустрічаються рідше. Ця ієрархічна схема кодування побудована шляхом рекурсивного поділу набору символів на дві підмножини на основі їх ймовірностей, поки кожна підмножина не міститиме один символ.

Алгоритм присвоює 0 підмножині з вищою ймовірністю та 1 підмножині з меншою ймовірністю. Процес повторюється для кожної підмножини, доки кожному символу не буде призначено унікальний код. У результаті утворюється префіксний код, який є типом коду змінної довжини, де жоден код не є префіксом будь-якого іншого коду.

Префіксний код, згенерований алгоритмом Шеннона-Фано, має унікальну властивість, необхідну для ефективного стиснення даних. Код завжди однозначний, тобто немає неоднозначності при декодуванні повідомлення з його стисненої форми. Це пояснюється тим, що жоден код не є префіксом будь-якого іншого коду, що дозволяє легко визначити, де закінчується один код і починається інший.

Для перетворення даного алгоритму із методу стиснення в метод шифрування достатньо в текст на основі якого будуть шифруватися повідомлення додати всі необхідні для спілкування символи що дозволить в словнику значень створити унікальні коди для всіх потрібних символів. Даний підхід сильно зменшить ефективність стиснення проте дозволить обмінюватися шифрованими повідомленнями для розшифрування яких буде потрібна знати вхідний текст на основі якого було виконано кодування.

2.5 Структура додатку

Розглянемо структурні особливості додатку за допомогою діаграми класів що спростить розуміння архітектури даного додатку та дасть розуміння особливостей реалізації багаторівневого шифрування.

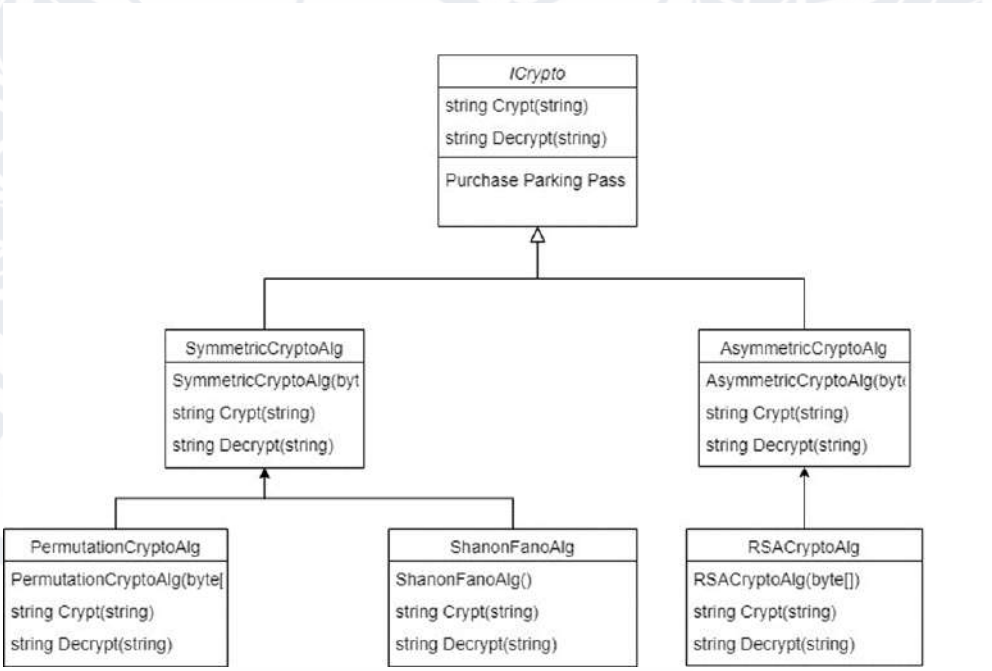

Рисунок 2.3 Діаграма класів криптографічної компоненти [50]

На рис. 2.3 наведено схему діаграми класів для реалізації криптографічних алгоритмів у месенджері з багаторівневим шифруванням. Вона складається з декількох класів та інтерфейсів:

ICrypto – основний інтерфейс, який містить методи що дозволяють здійснювати шифрування (Crypt) та дешифрування (Decrypt) повідомлень.

SymmetricCryptoAlg та AsymmetricCryptoAlg – абстрактні класи що забезпечують базове функціонування для симетричних та асиметричних криптографічних алгоритмів відповідно.

PermutationCryptoAlg – конкретна реалізація алгоритму перестановки для реалізації шифрування повідомлень на основі класу SymmetricCryptoAlg.

ShanonFanoAlg – реалізацією симетричного алгоритму шифрування яка використовує реалізовані класи та методи які безпосередньо не знаходяться в даному класі проте викликаються, обгортка реалізується на основі класу SymmetricCryptoAlg.

RSACryptoAlg – реалізація асиметричного алгоритму шифрування, на основі абстрактного класу AsymmetricCryptoAlg.

Симетричні алгоритми, такі як ті, що реалізовані в класі SymmetricCryptoAlg, використовують один і той самий ключ для шифрування та розшифрування даних. Цей клас містить методи Crypt та Decrypt, реалізовані відповідно до симетричних алгоритмів, а також додаткові методи, специфічні для цього типу алгоритмів.

З іншого боку, асиметричні алгоритми, представлені класом AsymmetricCryptoAlg, застосовують пару ключів: відкритий ключ для шифрування даних та закритий ключ для їх розшифрування. Цей підхід забезпечує більш високий рівень безпеки, особливо для передачі даних через незахищені канали зв'язку. Так само, як і SymmetricCryptoAlg, цей клас реалізує власні методи Crypt та Decrypt, а також додаткові методи, специфічні для асиметричних алгоритмів.

Розглянемо переваги даної архітектури.

Модульність та розширюваність: Завдяки ієрархічній структурі класів та використанню спадкування, додавання нових алгоритмів шифрування або модифікація існуючих стає відносно простим завданням. Це полегшує розширення функціональності системи в майбутньому.

Інкапсуляція: Кожен клас encapsulates свою реалізацію алгоритму шифрування, приховуючи деталі реалізації від інших частин системи. Це сприяє підтримці коду та його модифікації.

Повторне використання коду: Базові класи ICrypto, SymmetricCryptoAlg та AsymmetricCryptoAlg можуть бути повторно використані в інших проектах, що економить час та зусилля розробників.

Абстракція: Клас ICrypto забезпечує загальний інтерфейс для операцій шифрування та розшифрування, абстрагуючи ці операції від конкретних алгоритмів, що використовуються.

Зокрема для коректної роботи нам необхідно використовувати фабричні метод для створення конкретних реалізацій даних класів.

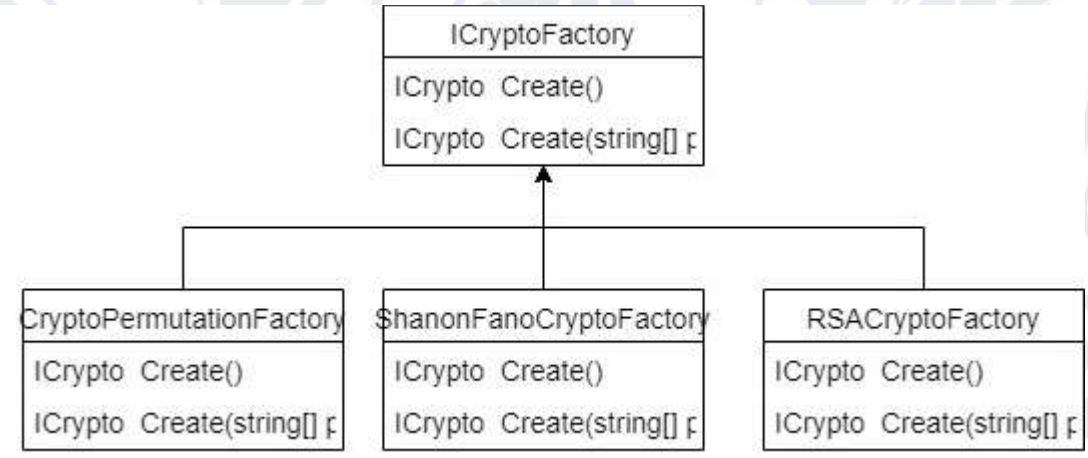

Рисунок 2.4 Діаграма фабричних класів [50].

На рисунку 2.4 наведено діаграму фабричних методів яка складається із таких класів як:

ICryptoFactory – визначає інтерфейс для створення криптографічних об'єктів через два методи:

ICrypto Create() – створює об'єкт без додаткових параметрів

ICrypto Create(string[] param) – створює об'єкт, приймаючи масив параметрів.

CryptoPermutationFactory – створює об'єкти, що реалізують криптографію з використанням перестановок.

ShanonFanoCryptoFactory – створює об'єкти, що реалізують криптографію з використанням методу Шеннона-Фано.

RSACryptoFactory – створює об'єкти, що реалізують RSA криптографію.

Таким чином, ця діаграма демонструє застосування шаблону Factory Method для створення різних видів криптографічних об'єктів через абстрактну фабрику та її конкретні реалізації.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ КРОСПЛАТФОРМНОГО МЕСЕНДЖЕРА

3.1 Структура клієнтської частини

3.1.1 Алгоритми методів шифрування

Алгоритм Шенона–Фано.

Алгоритм Шенона–Фано розпочинає свою роботу із створення екземпляра класу ShannonAlgorithm завдання якого полягає в створені коду для кодування повідомлень.

Лістинг 3.1.1 клас ShanonAlgorithm

```
internal class ShannonAlgorithm
{
    private Dictionary<char, string> LettersCoding = new
Dictionary<char, string>();
    private char[] letters;
    private int[] lettersCount;
    private bool flag = true;
    public ShannonAlgorithm(char[] letters, int[] lettersCount)
    {
        this.letters = letters;
        this.lettersCount = lettersCount;
        this.GetShannonCodeByLetters(' ', " ", 0, letters.Length -
1);
    }
    private void GetShannonCodeByLetters(char branch, string
full_branch, int startPos, int endPos){…}
    public Dictionary<char, string> GetLettersCode()
    {
        return LettersCoding;
    }
}
```

Даний клас складається із наступних полів та методів:

Поля класу:

LettersCoding: Словник (Dictionary) для зберігання закодованих значень для кожного символу. Ключем є сам символ, а значенням - його закодоване представлення.

letters: Масив символів, на основі яких буде створено шифр для подальшого шифрування повідомлень.

lettersCount: Масив цілих чисел, який містить кількість появ кожного символу у вхідних даних. Індекс у цьому масиві відповідає індексу символу в масиві letters.

flag: Логічна змінна, яка використовується в алгоритмі кодування для правильної роботи рекурсії із методу GetShannonCodeByLetters.

Методи класу:

Конструктор ShannonAlgorithm: Цей метод виконується при створенні нового об'єкта класу ShannonAlgorithm. Він отримує два масиви: letters, який містить символи, та lettersCount, що представляє кількість кожного символу у тексті. Конструктор ініціалізує внутрішні дані класу і викликає метод GetShannonCodeByLetters для обчислення кодів Шеннона для заданих символів.

Метод GetShannonCodeByLetters: Цей метод використовує рекурсивний підхід для обчислення кодів Шеннона для символів, які передаються йому у вигляді діапазону. Він приймає вхідні параметри, такі як символи branch, повний код full_branch та позиції startPos та endPos, які вказують на діапазон символів для обробки. Після обчислення кодів символів вказаного діапазону метод зберігає їх у словнику LettersCoding.

Метод GetLettersCode: Цей метод повертає результат обчислення кодів Шеннона для символів у формі словника, де символи є ключами, а їхні коди – значеннями. Він дозволяє отримати готовий результат для подальшого використання у програмі.

Проте перш ніж передати дані в даний клас їх потрібно підготувати для чого використовуються клас SortData оскільки він має необхідні методи які дозволять отримати відсортовані, підраховані та впорядковані масиви даних. Які в подальшому зможе використати клас ShanonAlgorithm.

Лістинг 3.1.2 клас SortData

```
internal class SortData
{
    private char[] data;
    private List<Letter> ListLetter = new List<Letter>() { };
    private double sum = 0;
    public SortData(char[] userInput)
    {…}
```

```
    private void GetFullSum()
    {…}
    public List<Letter> GetData()
    {…}
    private void SortInterest()
    {…}
    private void CheckArray(char elem)
    {…}
    public void Worker()
   {…}
}
```

Клас SortData складається із наступних полів та методів:

Поля класу:

data: масив символів, що містить вхідні дані та символи з Configuration.Language.

ListLetter: список об'єктів Letter, які представляють символи та їхню частоту появи.

sum: загальна кількість символів у вхідних даних.

Методи класу:

SortData(char[] userInput): конструктор класу, який приймає масив символів userInput та викликає метод Worker().

GetFullSum(): приватний метод, який обчислює загальну кількість символів, підсумовуючи частоту появи кожного символу в ListLetter.

GetData(): публічний метод, який повертає відсортований за зацікавленістю список ListLetter у зворотному порядку.

SortInterest(): приватний метод, який сортує елементи ListLetter за їхньою зацікавленістю (властивість interest класу Letter) за допомогою бульбашкового сортування.

CheckArray(char elem): приватний метод, який перевіряє, чи існує символ elem у ListLetter. Якщо існує, збільшує його частоту появи на 1. Якщо не існує, додає новий об'єкт Letter з цим символом та частотою появи 1 до ListLetter.

Worker(): публічний метод, який виконує основну логіку класу. Він проходить по кожному символу у data, викликаючи CheckArray(char elem). Потім обчислює загальну кількість символів за допомогою GetFullSum(), встановлює значення зацікавленості для кожного символу в ListLetter за допомогою методу

SetInterest(double sum) класу Letter та, нарешті, сортує ListLetter за зацікавленістю за допомогою SortInterest().

Для роботи класу SortData використовується такий клас як Letter який є об’єктом для збереження додаткової інформації для виконання алгоритму.

Лістинг 3.1.3 класу Letter

```
internal class Letter
{
    public char letter;
    public int count;
    public double interest;
    public string letterCode = "";
    public Letter(char letter, int count)
    {
        this.letter = letter;
        this.count = count;
    }
    public void SetInterest(double all)
    {
        interest = Math.Round((count / all) * 100.0, 3);
    }
}
```

Клас Letter складається із наступних полів та методів:

Поля:

letter (тип char): поле, яке зберігає символьне значення літери.

count (тип int): поле, яке зберігає кількість входжень цієї літери.

interest (тип double): поле, яке зберігає відсоток входжень цієї літери по відношенню до загальної кількості.

letterCode (тип string): рядкове поле, яке використовується для зберігання коду літери.

Методи:

Letter(char letter, int count): конструктор класу, який приймає символьне значення літери та її кількість, і ініціалізує відповідні поля letter та count.

SetInterest(double all): метод, який обчислює відсоток входжень цієї літери по відношенню до загальної кількості all. Відсоток обчислюється шляхом ділення count на all, множення на 100, а потім округлення до трьох десяткових знаків за допомогою Math.Round(). Результат зберігається у полі interest.

Загалом сукупність класів ShanonAlgorithm, SortData та Letter виконують функцію створення словника (значення, ключ) для кодування повідомлень після чого метод для обробки повідомлення замінює літери на код.

Для декодування повідомлень було створено клас Decoding

Лістинг 3.1.4 клас Decoding

```
internal class Decoding
{
    private int GetMaxLenValue()
    {…}
    public string GetResult(string userInput)
    {…}
}
```

Клас Decoding складається із наступних методів:

Метод GetMaxLenValue():

Цей метод знаходить найдовший двійковий код серед усіх кодів у словнику Configuration.letterCodeValues. Після чого він перебирає всі коди в словнику і запам'ятовує довжину найдовшого коду в кінці повертаючи довжину найдовшого коду.

Метод GetResult(string userInput):

Цей метод приймає рядок userInput, який складається з двійкового коду, і перетворює його на рядок символів.

Спочатку метод перевіряє, чи увесь userInput збігається з одним з кодів у словнику. Якщо так, він повертає відповідний символ.

Якщо userInput складається з декількох кодів, метод розбиває його на окремі коди і декодує їх один за одним.

Він зчитує перший код із userInput, знаходить відповідний символ у словнику і додає його до результату.

Потім метод зчитує наступний код із userInput, знаходить відповідний символ і додає його до результату.

Цей процес продовжується, доки не буде декодований увесь userInput.

Після декодування всіх кодів метод повертає результуючий рядок символів.

Загалом взаємодія цих класів реалізує алгоритм Шенона–Фано та виконує всі необхідні дії для подальшого використання його у вигляді методу шифрування повідомлень.

Метод шифрування RSA.

Метод шифрування RSA було додано в застосунок за рахунок використання простору імен System.Security.Cryptography всередині платформи .NET тож розглянемо основні поля та методи даного класу [51].

Create(): Створює новий екземпляр класу RSA за замовчуванням.

ImportParameters(RSAParameters): Імпортує параметри ключа RSA з об'єкту RSAParameters.

ExportParameters(bool): Експортує параметри ключа RSA до об'єкту RSAParameters. Boolean параметр вказує, чи експортувати відкритий (false) чи приватний (true) ключ.

Encrypt(byte[], RSAEncryptionPadding): Шифрує вхідні дані за допомогою публічного ключа. Потрібно вказати об'єкт RSAEncryptionPadding для обрання схеми доповнення.

Decrypt(byte[], RSAEncryptionPadding): Розшифровує вхідні дані за допомогою приватного ключа. Потрібно вказати об'єкт RSAEncryptionPadding для обрання схеми доповнення.

SignData(byte[], HashAlgorithmName, RSASignaturePadding): Створює цифровий підпис для вхідних даних за допомогою приватного ключа, використовуючи вказані алгоритм хешування та схему доповнення.

VerifyData(byte[], byte[], HashAlgorithmName, RSASignaturePadding): Перевіряє цифровий підпис для вхідних даних за допомогою публічного ключа, використовуючи вказані алгоритм хешування та схему доповнення.

Шифрування методом перестановки.

Даний метод шифрування працює по принципу заміни символів шляхом перестановки їх місцями на задану кількість символів та реалізований класом PermutationMethod.

Загалом, цей клас PermutationMethod реалізує простий метод шифрування - перестановку символів у рядку. Ідея полягає в тому, щоб змінити порядок символів відповідно до певного ключа (в даному випадку – permutationNumber), щоб зробити вихідний текст нечитабельним. Алгоритми шифрування та дешифрування є оберненими операціями.

Лістинг класу PermutationMethod

```
public class PermutationMethod : MessageHandler
{
    public string EncryptMessage(string message)
    {…}
    public string DecryptMessage(string message)
    {…}
}
```

Клас PermutationMethod складається із наступних методів:

EncryptMessage приймає вхідний рядок для шифрування. Отримує значення ключа (permutationNumber) з класу Configuration. Перетворює вхідний рядок на масив символів. Для кожного символу в масиві обчислює новий індекс за формулою, що залежить від поточного індексу, ключа та довжини рядка. Міняє місцями поточний символ з символом на новому індексі. Повертає новий рядок, створений з перемішаного масиву символів.

DecryptMessage працює аналогічно, але у зворотному порядку, використовуючи той самий ключ для відновлення вихідного порядку символів.

Хоч даний метод шифрування є не дуже складним проте якщо його використовувати в поєднанні із методом Шенона–Фано він зможе значно ускладнити дешифрування повідомлення для потенційного зловмисника або ж зацікавленої особи якій потрібна інформація із повідомлень.

3.1.2 Інтерфейс користувача

Розглянемо інтерфейс нашого месенджера та познайомимося із його можливостями шляхом огляду елементів інтерфейсу.

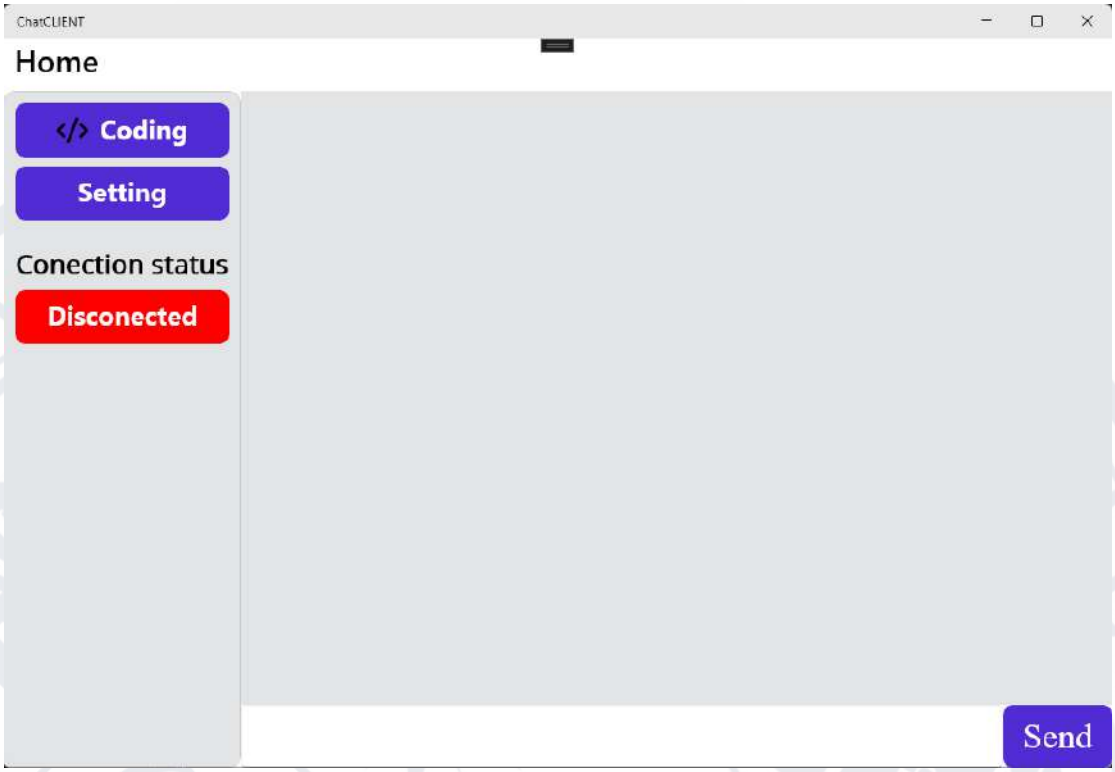

Рисунок 3.1 – Головна сторінка додатку

На сам перед голосну сторінку можна розділити на дві функціональні панелі ліва панель що містить кнопки Coding, Setting та кнопка підключення до сервера що знаходиться під надписом Connection status при взаємодії із нею вона змінює надпис із Disconected на Conected в свою чергу вказуючи на те чи підключений користувач до сервера чи ні.

Кнопка Coding переводить на меню для створення шифру за методом Шеннона–Фано пізніше розглянемо дане меню більш детально.

Кнопка Setting відповідає за перехід на сторінку налаштування застосунку в якій можна обрати ті чи інші налаштування або ж змінити функціонал застосунку шляхом зміни додаткового метода шифрування.

Права частина застосунку відповідає за написання, відправку та відображення повідомлень також для кращого розуміння повідомлення написані користувачем будуть відображатися в правій частині даної панелі а повідомлення отримані від сервера будуть відображатися в лівій частині екрану для підвищення розуміння. Додатково повідомлення написані користувачем

будуть відображатися кольором який в форматі hex записується як #002451, повідомлення від сервера будуть виділятися кольором який в форматі hex записується як #A9E0B9.

Для створення повідомлень використовується такий елемент як Entry який дозволяє вводити, редагувати та переглядати повідомлення.

Дещо правіше даного поля для вводу повідомлень знаходиться кнопка Send яка додає повідомлення на елемент ScrollView який надає змогу розмістити на ньому необмежену кількість повідомлень за рахунок того що дозволяє переглядати всі надіслані повідомлення та відправляє його на сервер через протокол TSP.

Загалом сукупність цих елементів візуалізації та взаємодії формують представлення головної сторінки месенджера.

Розглянемо сторінку для створення шифру за алгоритмом Шеннона–Фано.

Рисунок 3.2 – Сторінка кодування.

На рисунку 3.2 під номером 1 знаходиться кнопка для повернення на головну сторінку.

Під номером 2 поле для вводу тексту на основі якого буде створено шифр використовуючи алгоритм Шеннона–Фано.

Під номером 3 знаходиться кнопка для створення коду із введеного в поле 2 тексту вона відповідає за передачу в клас ShanonAlgorithm який в свою чергу створює словник кодування повідомлень.

При коректно ведених даних надпис під номером 4 змінює своє значення на Completed. У випадку некоректно веденого тексту зявляється попереджувальне повідомлення яке викликається за рахунок методу DisplayAlert().

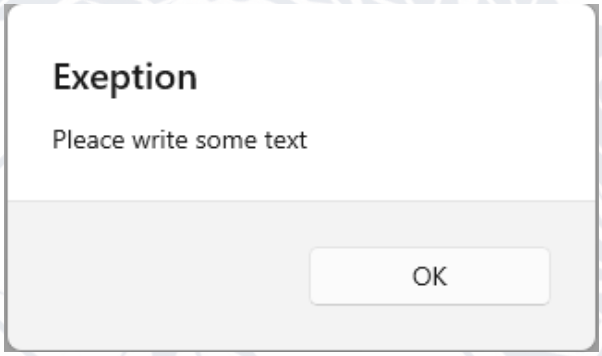

Рисунок 3.3 – Попереджувальне повідомлення

Для коректного обміну повідомлень користувачі які бажають спілкуватися один з одним повинні використовувати однакові шифри а отже текст написаний ними в елемент під номером 2 повинні бути ідентичними.

Загально робота даних елементів дає змогу створювати словник шифрування з використанням алгоритму Шеннона–Фано для подальшого використання цього словника для шифрування повідомлень.

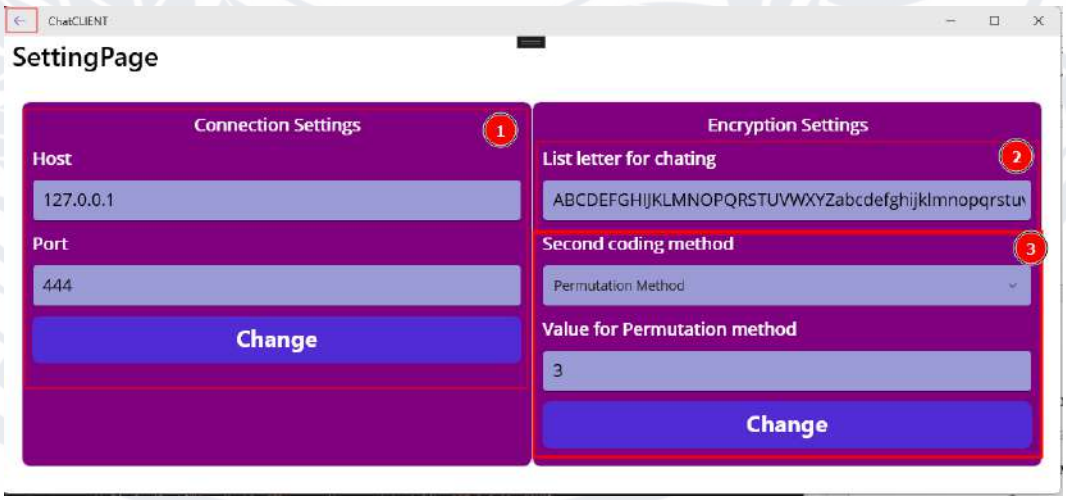

Рисунок 3.4 – Сторінка налаштування

На рисунку 3.4 продемонстровано сторінку налаштувань яка виконує функціонал пов'язаний із методами шифрування та інформація для підключення до сервера по TCP протоколу.

Елемент під номером 1 із рисунка 3.4 представляє собою поля для вводу host та port по яких буде знаходитися сервер для обміну повідомленнями.

Елемент під номером 2 представляє поле для вводу символів які буде використовувати користувач для спілкування програмно для цього поля встановлено значення яке дорівнює всім літерам українського та англійського алфавіту та всі прості числа з окремими додатковими символами.

Значення даного поля при запуску програми дорівнює:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzАБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯабвгґдеєжзиіїйклмнопрстуфхцчшщьюя1234567890!?,.`

Дані символи представляють повний набір основних символів в спілкуванні тому його має бути достатньо для обміну повідомленням при потребі можна додати своїх символів або ж скоротити кількість символів, наприклад прибравши англійський алфавіт.

Під номером 3 на зображенні 3.4 знаходиться меню вибору додаткового методу шифрування та поле для вводу ключа для методу перестановки та кнопка яка застосовує зміни які будуть зроблені в даному полі.

Також для того щоб продемонструвати кросплатформність для даного застосунку, ми використаємо функціонал Visual Studio для збірки даного застосунку на Android пристроях. Це дозволить нам перевірити, чи працює наш застосунок належним чином на різних платформах, що є важливою вимогою для сучасних додатків.

Крім того, Visual Studio дозволяє легко налаштувати параметри збірки, такі як версія Android SDK, орієнтація екрану, необхідні дозволи та інші опції, що забезпечує гнучкість та зручність розробки крос-платформених застосунків. Таким чином, ми продемонструємо, що наш додаток може бути успішно

розгорнутий не лише на Windows, але й на Android пристроях, розширюючи його доступність та охоплення користувачів.

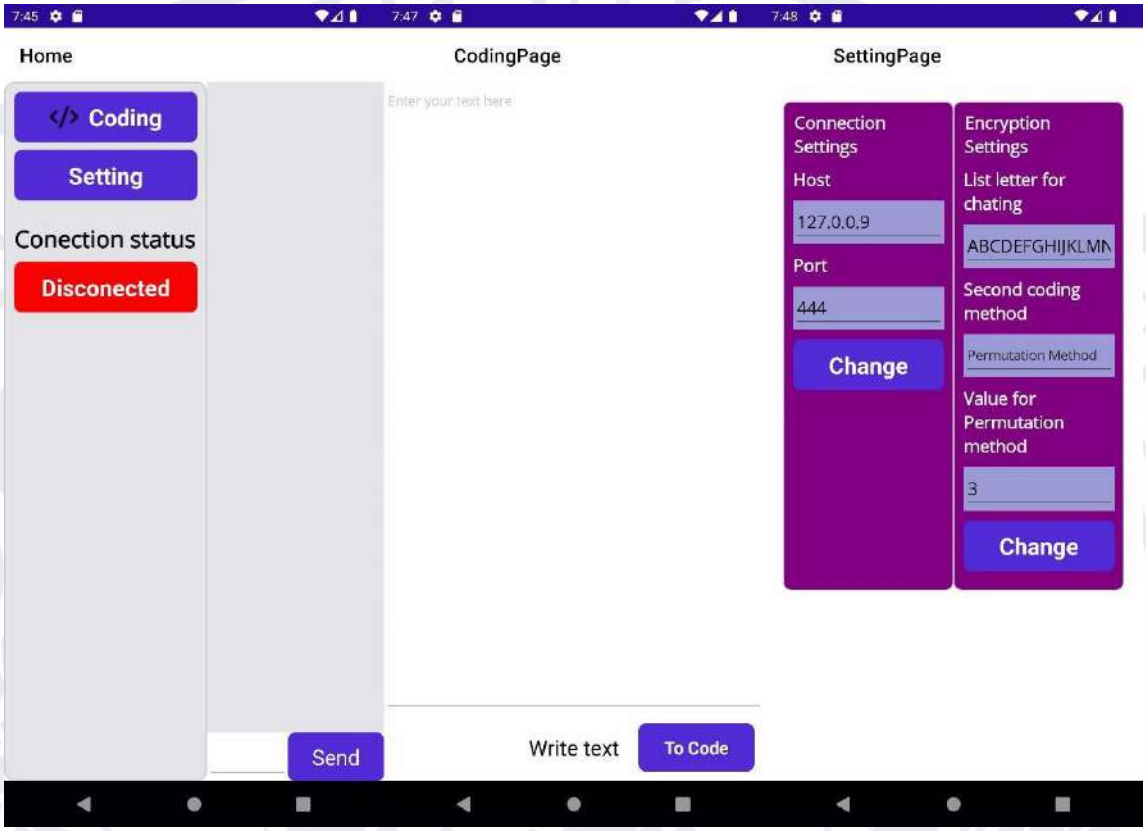

Рисунок 3.5 – Інтерфейс користувача на телефоні

Як видно із рисунку 3.5 інтерфейс користувача зберігає свій вигляд і функціонал на Android застосунках.

3.1.3 Способи взаємодії із сервером

Для взаємодії із сервером застосунок використовує клас SocketClient який реалізує певний набір функцій взаємодії із сервером.

Лістинг класу SockerClient

```
class SocketClient
{
    private TcpClient client;
    private const int receiveChunkSize = 1024;
    private networkStream stream;

    public event Action<string> MessageReceived;
    public SocketClient()
    {…}
    public async Task ConnectAsync(string host, int port)
```

```
    {…}
    private async Task ReceiveMessagesAsync()
    {…}
    public async Task SendMessageAsync(string message)
    {…}
    public async Task CloseAsync()
    {…}
}
```

Даний клас складається із наступних полів та методів:

Поля класу:

client: Об'єкт класу TcpClient, який представляє клієнтський сокет TCP.

receiveChunkSize: Константа, яка визначає розмір буферу для прийому повідомлень у байтах.

stream: Об'єкт класу NetworkStream, який використовується для читання та запису даних через сокет.

ressageReceived: Подія, яка викликається, коли приймається нове повідомлення від сервера.

Методи класу:

SocketClient(): Конструктор класу, який ініціалізує об'єкт TcpClient.

ConnectAsync(string host, int port): Метод, який асинхронно встановлює підключення до сервера за заданим хостом та портом. Після встановлення підключення, він відправляє ім'я користувача на сервер та запускає задачу для прийому повідомлень від сервера.

ReceiveMessagesAsync(): Приватний метод, який асинхронно читає дані з потоку stream та викликає подію MessageReceived для кожного отриманого повідомлення.

SendMessageAsync(string message): Метод, який асинхронно відправляє повідомлення на сервер, якщо клієнт підключений.

CloseAsync(): Метод, який асинхронно закриває підключення клієнта до сервера та виводить повідомлення у консоль.

Отже даний клас реалізує необхідний набір інструментів для взаємодії із сервером та забезпечує зв'язок із сервером через TCP клієнт при цьому не впливаючи на роботу програми оскільки всі його методи працюють асинхронно.

3.2 Структура серверної частини

Серверна частина застосунку призначена для з'єднання клієнтів між собою за допомогою TCP протоколу реалізована у вигляді консольного застосунку .

Серверна частина складається із таких класів як Program, ClientObject та ServerObject робота сервера розпочинається із класу Program який створює та в окремому потоці запускає роботу сервера шляхом створення обєкта типу SevrverObject. Розглянемо клас ServerObject та які методи він має.

Лістинг класу ServerObject

```
internal class ServerObject
{
    static TcpListener tcpListener;
    List<ClientObject> clients = new List<ClientObject>();
    private string port;
    private string host;
    public ServerObject(string port, string host)
    {…}
    protected internal void AddConnection(ClientObject
clientObject)
    {…}
    protected internal void RemoveConnection(string id)
    {…}
    protected internal void Listen()
    {…}
    protected internal void BroadcastMessage(string message,
string id = "0")
    {…}
    protected internal void Disconnect()
    {…}
}
```

Даний клас складається із наступних полів та методів:

Поля класу:

tcpListener (static TcpListener): Об'єкт класу TcpListener, який є частиною .NET Framework і використовується для прослуховування вхідних TCP-з'єднань.

clients (List<ClientObject>): Список об'єктів класу ClientObject, які представляють клієнтів, підключених до сервера.

port (string): Рядок, який містить номер порту, на якому сервер буде очікувати з'єднання.

host (string): Рядок, який містить IP-адресу сервера.

Методи класу:

ServerObject(string port, string host): Конструктор, який приймає номер порту та IP-адресу як параметри і присвоює їх відповідним полям.

AddConnection(ClientObject clientObject) (protected internal): Цей метод додає об'єкт ClientObject до списку clients.

RemoveConnection(string id) (protected internal): Цей метод видаляє об'єкт ClientObject зі списку clients за його ідентифікатором id.

Listen() (protected internal): Цей метод запускає прослуховування вхідних TCP-з'єднань на заданому порту та IP-адресі. Коли клієнт підключається, він створює новий об'єкт ClientObject та запускає новий потік для обробки з'єднання. Метод перехоплює виключення та викликає Disconnect() у блоці finally.

BroadcastMessage(string message, string id = "0") (protected internal): Цей метод відправляє повідомлення message всім підключеним клієнтам, крім клієнта з ідентифікатором id.

Disconnect() (protected internal): Цей метод зупиняє прослуховування TCP-з'єднань, закриває всі з'єднання з клієнтами та завершує виконання програми.

Даний клас активно використовує додатковий клас ClientObject який представляє клієнта розглянемо більш детально даний клас.

Лістинг класу ClientObject

```
internal class ClientObject
{
    protected internal string Id { get; private set; }
    protected internal NetworkStream Stream { get; private set; }
    TcpClient client;

    public ClientObject(TcpClient tcpClient, ServerObject
serverObject)
    {…}
    public void Process()
    {…}
    private string GetMessage()
    {…}
    protected internal void Close()
    {}
}
```

Даний клас складається із наступних полів та методів:

Поля:

Id (string): Властивість, яка містить унікальний ідентифікатор з'єднання з клієнтом.

Stream (NetworkStream): Властивість, яка представляє потік даних для обміну даними з клієнтом.

client (TcpClient): Властивість, яка зберігає об’єкт класу TcpClient який містить методи необхідні для відправки та отримання повідомлень

Методи:

Process(): Цей метод є основним методом для обробки з'єднання з клієнтом. Він отримує потік даних з клієнта, отримує вхідні повідомлення та відправляє їх на сервер для розсилки іншим клієнтам. Метод працює в циклі, поки клієнт не відключиться або не виникне виключення. Якщо з'єднання втрачено, метод видаляє поточний об'єкт ClientObject зі списку клієнтів на сервері та закриває з'єднання.

GetMessage() (private): Цей метод читає вхідні дані з потоку даних клієнта та повертає отримане повідомлення як рядок.

Close() (protected internal): Цей метод закриває потік даних та підключення до клієнта.

Взаємодія даних класів між собою забезпечує роботу сервера до якого може підключатися наш клієнтський застосунок та здійснювати обмін повідомленнями в реальному часі.

3.3 Реалізація багаторівневого шифрування повідомлень

Для реалізації багаторівневого шифрування було створено інтерфейс MessageHandler який узагальнює способи використання методів шифрування шляхом створення похідних класів які реалізують даний інтерфейс.

Лістинг інтрфейсу MessageHandler

```
public interface MessageHandler
{

    string EncryptMessage(string message);
    string DMessage(string encryptMessage);

}
```

Реалізація даного інтерфейсу для кожного із методів дозволяє бути впевнених що якщо методи дешифрування будуть прописані в зворотному порядку до методів шифрування то повідомлення буде розшифроване коректно.

Розгляньмо приклад роботи застосунку які однаково налаштовані.

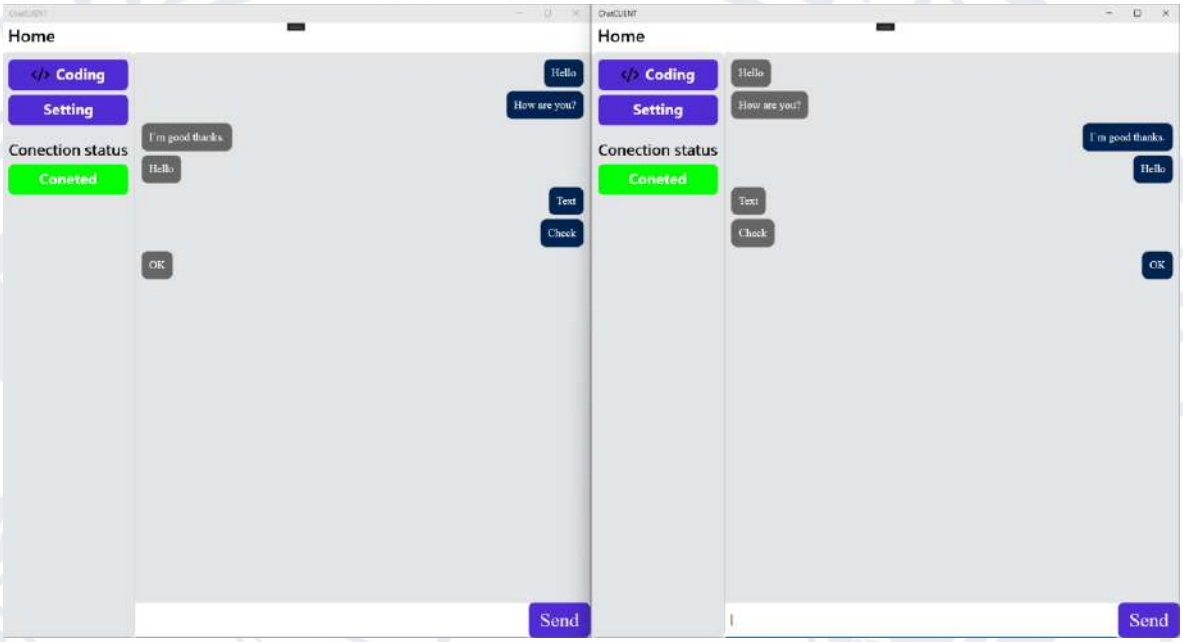

Рисунок 3.6 – Приклад роботи застосунку

Змінимо налаштування одного з клієнтів одного з клієнтів щоб імітувати підключення не санкціонованого клієнта.

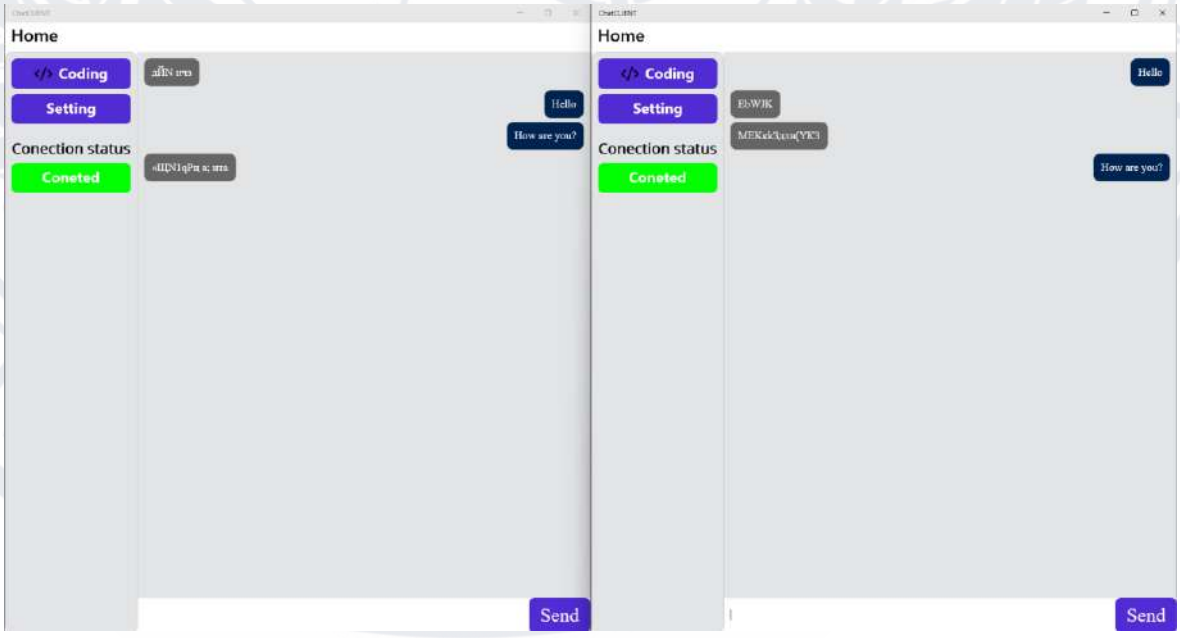

Рисунок 3.7 – Приклад роботи застосунку із різними налаштуваннями

Як видно із рисунку 3.7 при різних налаштуваннях шифрування спроба спілкування стає неможливою. А тепер повернемо в них правильні налаштування.

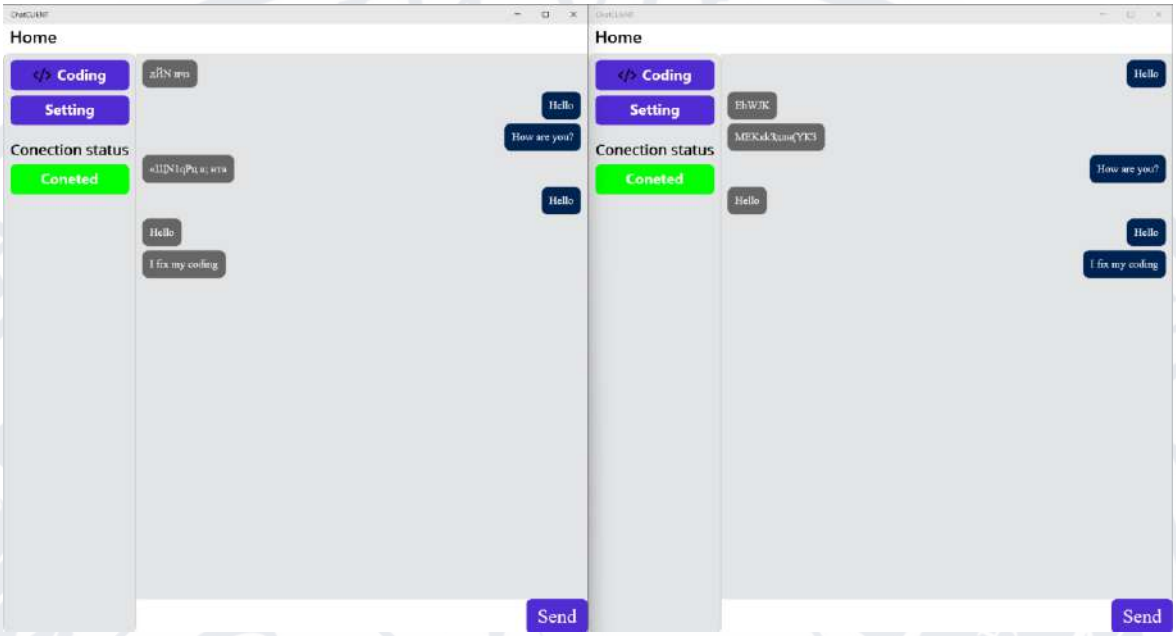

Рисунок 3.8 – Робота застосунку після зміни налаштувань на ідентичні.

Як видно із рисунку 3.8 після встановлення ідентичних налаштувань шифрування в обох клієнтах спілкування стає можливим. Що свідчить про коректність роботи методів шифрування.

Загалом коректна робота методів шифрування свідчить про правильність архітектури для комбінації методів шифрування що забезпечує високий рівень шифрування.

ВИСНОВКИ

У результаті виконання даної кваліфікаційної роботи було розроблено безпечний кросплатформний месенджер з багаторівневим шифруванням повідомлень та багатошаровою архітектурою.

Під час аналізу предметної області було розглянуто актуальні платформи, кросплатформний підхід до розробки додатків, особливості розробки клієнт-серверних програм, багатошарову архітектуру та методи шифрування повідомлень. Також було сформовано технічне завдання, проаналізовано архітектурні та проектувальні патерни, обрано стек технологій для реалізації та визначено структуру додатку.

Під час реалізації було створено клієнтську та серверну частини месенджера. На клієнтській стороні реалізовано алгоритми шифрування, інтерфейс користувача та способи взаємодії з сервером. Серверна частина відповідає за обробку та маршрутизацію шифрованих повідомлень між клієнтами.

Ключовою особливістю розробленого месенджера є реалізація багаторівневого шифрування повідомлень з використанням комбінації різних алгоритмів, таких як метод перестановки, RSA та Шеннона-Фано. Це забезпечує високий рівень захисту конфіденційної інформації та ускладнює її дешифрування зловмисниками.

Отже, розроблений месенджер з багаторівневим шифруванням повідомлень та багатошаровою архітектурою забезпечує надійний захист конфіденційних комунікацій, що є критично важливим у сучасному цифровому середовищі з постійно зростаючими кіберзагрозами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Найпопулярніші операційні системи в світі URL: https://marketer.ua/ua/stats-operating-system-2017/ (дата звернення: 10.03.2024)
2. Android. URL: https://uk.wikipedia.org/wiki/Android (дата звернення: 10.03.2024)
3. Brian H., Bill P. Android Programming: The Big Nerd Ranch Guide" (4th Edition), 2019. 832 с.
4. IOS. URL: https://uk.wikipedia.org/wiki/IOS (дата звернення: 10.03.2024)
5. Wallace W. Beginning iPhone Development with Swift 5: Exploring the iOS SDK, 2019. 786 с.
6. MacOS. URL: https://uk.wikipedia.org/wiki/MacOS (дата звернення: 10.03.2024)
7. Sagar R., Jasdeep S. Exploring macOS: A Journey Through the Mac Ecosystem, 2023. 365 с.
8. Herta N. The macOS User Administration Guide, 2021. 804 с.
9. Windows URL: https://uk.wikipedia.org/wiki/Microsoft_Windows (дата звернення: 10.03.2024)
10. Ed B. Windows 11 Inside Out, 2023. 816 с.
11. Michael H. Learning Modern Linux, 2022. 260 с.
12. Marshall Kirk McKusick, George V. Neville-Neil, Robert N.M. Watson Design and Implementation of the FreeBSD Operating System, 2nd Edition, 2014, 928 с.
13. QNX URL: https://uk.wikipedia.org/wiki/QNX (дата звернення: 10.03.2024)
14. Cross-platform software. URL: https://en.wikipedia.org/wiki/Cross-platform_software (дата звернення: 11.03.2024)
15. Mark J. Price C# 12 and .NET 8 – Modern Cross-Platform Development Fundamentals: Start building websites and services with ASP.NET Core 8, Blazor, and EF Core 8, 2018, 828 с.
16. Офіційна документація React Native. URL: https://reactnative.dev/docs/getting-started (дата звернення: 15.03.2024)
17. Mikhail S., Adam B. React and React Native - Fifth Edition, 2024, 508 с.
18. Офіційна документація Flutter. URL: https://flutter.dev/docs (дата звернення: 15.03.2024)
19. Thomas B., Alessandro B. Flutter for Beginners - Third Edition, 2023, 406 с.
20. Офіційна документація Xamarin. URL: https://learn.microsoft.com/en-us/xamarin/ (дата звернення: 15.03.2024)
21. Daniel H., Johan K. Xamarin.Forms Projects - Second Edition, 2020, 504 с.
22. Офіційна документація Apache Cordova. URL: https://cordova.apache.org/docs/en/12.x/ (дата звернення: 15.03.2024)

23. Офіційна документація Ionic URL: https://ionic.io/docs (дата звернення: 16.03.2024)
24. Chris G. Mobile App Development with Ionic, Revised Edition, 2017, 290 с.
25. Офіційна документація Qt URL: https://doc.qt.io/ (дата звертання: 17.03.2024)
26. Nibedit D. Cross-Platform Development with Qt 6 and Modern C++, 2021, 442 с.
27. Офіційна документація MAUI URL: https://learn.microsoft.com/en-us/dotnet/maui/?view=net-maui-8.0 (дата звертання: 18.03.2024)
28. Matthew Goldman .NET MAUI in Action, 2023, 456 с.
29. Michael C., Daniel H., Johan K. .NET MAUI Projects - Third Edition, 2024, 630 с.
30. Jesse L., Rodrigo J. .NET MAUI for C# Developers, 2023, 296 с.
31. Клієнт-серверна архітектура URL: https://uk.wikipedia.org/wiki/Клієнт-серверна_архітектура (дата звертання: 10.04.2024)
32. James S. Deciphering Data Architectures, 2024, 419 с.
33. Raju G., Mark R., Neal F. Head First Software Architecture, 2024, 486 с.
34. David W. Real-World Cryptography, 2021, 400 с.
35. Шифрування URL: https://uk.wikipedia.org/wiki/Шифрування (дата звертання: 11.04.2024)
36. Model-View-ViewModel URL: https://uk.wikipedia.org/wiki/Model-View-ViewModel (дата звертання: 16.04.2024)
37. Pieter N. The MVVM Pattern in .NET MAUI, 2023, 386 с.
38. Mark J. Price C# 12 and .NET 8 – Modern Cross-Platform Development Fundamentals - Eighth Edition, 2023, 826 с.
39. Jason A. Clean Code with C# - Second Edition, 2023, 492 с.
40. Jort R. Code like a Pro in C#, 2021, 416 с.
41. Gerard B. Target C#: Simple Hands-On Programming with Visual Studio 2022, 2022, 1084 с.
42. XAML Документація URL: https://www.noesisengine.com/docs/Gui.Core.XamlIntroduction.html (дата звертання: 28.04.2024)
43. Dan H., Nima M. Building Xamarin.Forms Mobile Apps Using XAML: Mobile Cross-Platform XAML and Xamarin.Forms Fundamentals,2019, 445 с.
44. Anna S. Learning Git, 2023, 317 с.
45. Prem Kumar Ponuthorai, Jon L. Version Control with Git, 3rd Edition, 2022, 546 с.
46. TCP Протокол URL: https://uk.wikipedia.org/wiki/TCP (дата звертання: 30.04.2024)
47. Joe C. Sams Teach Yourself TCP/IP in 24 Hours, 2017, 544 с.
48. RSA шифрування URL: https://uk.wikipedia.org/wiki/RSA (дата звернення 01.05.2024)

49. Перестановочний шифр URL: https://uk.wikipedia.org/wiki/Перестановочний_шифр (дата звернення 02.05.2024)

50. Оврамець І. В., Антонов Ю. С., Архітектурні особливості месенджера із багаторівневим шифруванням. Матеріали V Всеукраїнської науково-практичної конференції «Прикладні інформаційні технології». Вінниця, ДонНУ імені Василя Стуса, 2024.

51. RSA in .NET URL: https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.rsa?view=net-8.0 (дата звернення 20.05.2024)

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)