Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Ілюшина Олександра Сергійовича*
(ПІБ)

академічної групи *122-20-ск1*
(шифр)

спеціальності *122 Комп'ютерні науки та інформаційні технології*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки та інформаційні технології*
(назва освітньої програми)

на тему: *Розробка кросплатформного додатку для організації чату з використанням .NET та мови C#*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Кабак Л.В.*			
розділів:				
спеціальний	*доц. Кабак Л.В.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-20-ск1* (група) *Ілюшина Олександра Сергійовича* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка кросплатформного додатку для організації чату з використанням використанням .NET та мови C#*

затверджена наказом ректора НТУ «ДП» від *21.05.2023 р.* № *770-л*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2020 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПО й тривалості його розробки***	***27.05.2020 р.***

Завдання видав _____ (підпис) *доц. Кабак Л.В.* (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ (підпис) *Мухтарян М.А.* (прізвище, ініціали)

Дата видачі завдання: *р.*

Термін подання кваліфікаційної роботи до ЕК: *р.*

РЕФЕРАТ

Пояснювальна записка: 106 с., 28 рис., 5 дод., 10 джерел.

Об'єкт розробки: кросплатформний застосунок для організації чату.

Мета дипломного проекту: забезпечити зручний інструментарій для створення та використання чатів на різних платформах, всі можливі CRUD-операції для користувача, приватного, групового чатів та їх повідомлень.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної області, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування системи, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження застосунку, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної підсистеми, проведений підрахунок вартості роботи по створенню застосунку та розраховано час на його створення.

Актуальність даного програмного забезпечення визначається необхідністю користувачів швидко та ефективно обмінюватися інформацією на відстані.

Список ключових слів: КРОСПЛАТФОРМНИЙ ДОДАТОК, .NET, MAUI, C#, XAML, SQL SERVER, ENTITY FRAMEWORK CORE

ABSTRACT

Explanatory note: 106 pages, 28 pics, 5 apps, 10 sources.

Object of development: a cross-platform application for organizing chats.

The purpose of the thesis project: to provide a convenient toolkit for creating and using chats on different platforms, all possible CRUD operations for user, private, group chats and their messages.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the field of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes the existing solutions, selects the platform for development, performs design and development of the program, describes the algorithm and structure of the system, determines the input and output data, provides the characteristics of the parameters of hardware, describes the call and application load, describes the program.

In the economic section, the complexity of the developed information subsystem is determined, the cost of work on creating the application is calculated and the time for its creation is calculated.

The relevance of this software is determined by the need for users to quickly and efficiently exchange information at a distance.

List of keywords: CROSS-PLATFORM APPLICATION, .NET, MAUI, C#, XAML, SQL SERVER, ENTITY FRAMEWORK CORE

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

ЕОМ – електронно-обчислювальна машина.

CRUD – create, read, update, delete.

API – інтерфейс програмування додатків.

XAML — eXtensible Application Markup Language

ORM — Object-relation model

MAUI — Multi-platform App UI

ВСТУП

Сучасний світ перетворився на онлайн-середовище, і якість обміну інформацією важко переоцінити. На цьому тлі, наголошується на актуальності створення та підтримки веб-застосунків, а особливо крос-платформних застосунків, які дають змогу користувачам обмінюватися інформацією з будь-якого пристрою і з будь-якої точки світу.

Ця потреба яскраво виражена в нинішній глобалізованій економіці, але у зв'язку з пандемією і зростаючою потребою у віддаленій роботі, глобальний попит на діджиталізацію, айті-продукти і, зокрема, засоби комунікації зріс експоненціально. Чатові, аудіо-візуальні та файлові засоби комунікації стали необхідним як робочим робочим інструментом, так і інструментом пошуку цієї роботи

Створення крос-платформних веб-додатків стало важливим рішенням для забезпечення зв'язку та обміну інформацією між користувачами з різних пристроїв і з різних куточків світу. Ця потреба особливо гостро стала проявлятися в сучасному глобалізованому світі, де віддалена робота і цифровізація стали невід'ємною частиною життя.

Попит на ІТ-продукти та інструменти комунікації, як-от чати, аудіо- та відео-конференції, та засоби обміну файлами, зріс експоненціально, особливо у світлі пандемії, коли віддалена робота стала нормою для багатьох компаній. Ці інструменти стали не тільки необхідним робочим інструментом, а й засобом для пошуку роботи та комунікації з колегами, друзями та близькими людьми.

Створення крос-платформних веб-застосунків має велике значення, оскільки вони забезпечують зручний та ефективний обмін інформацією між користувачами, які працюють на різних пристроях і з різних місць. Такі додатки спрощують життя людям і допомагають їм залишатися пов'язаними, незважаючи на відстань.

Крім того, крос-платформні веб-додатки мають низку переваг перед традиційними додатками, які не можуть працювати на різних платформах. Вони забезпечують більшу гнучкість, дають змогу користувачам працювати на будь-якому пристрої, їхні дані зберігаються в хмарі, що робить їх доступними з будь-якої точки світу.

Усе це зумовлює підвищений інтерес до розроблення та використання крос-платформних веб-додатків. На сьогодні існує безліч інструментів і технологій, які дають змогу розробляти якісні крос-платформні застосунки. Такі додатки можуть бути використані в різних галузях, від бізнесу і маркетингу до освіти і медицини.

З перелічених вище міркувань я обрав своєю темою кваліфікаційної роботи «Розробка крос платформного додатку для організації чату»

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості с предметної галузі

Веб-чат (Web Chat) - це онлайн-сервіс, що дає змогу спілкуватися між користувачами через інтернет. Він дає змогу обмінюватися повідомленнями в режимі реального часу, а також передавати файли і проводити аудіо- та відео-дзвінки. Веб-чати широко використовуються в бізнесі, освіті, медицині та особистому житті для спілкування та обміну інформацією.

Історія веб-чат додатків сягає своїм корінням у далекі 1970-ті роки. Першим таким сервісом, а заодно і початком інтернету, був заснований у 1969 році **Arpanet** (англ. *Advanced Research Projects Agency Network* — Мережа передових досліджень), що була створена за дорученням Міністерства Оборони США та за участю наукових установ з метою об'єднання науково-дослідних і військових інститутів в одну мережу для поліпшення обміну інформацією між ними. У 1971 році досягнуто першочергової цілі по об'єднанню військових із науково-дослідницькими закладами(див. рис.1).

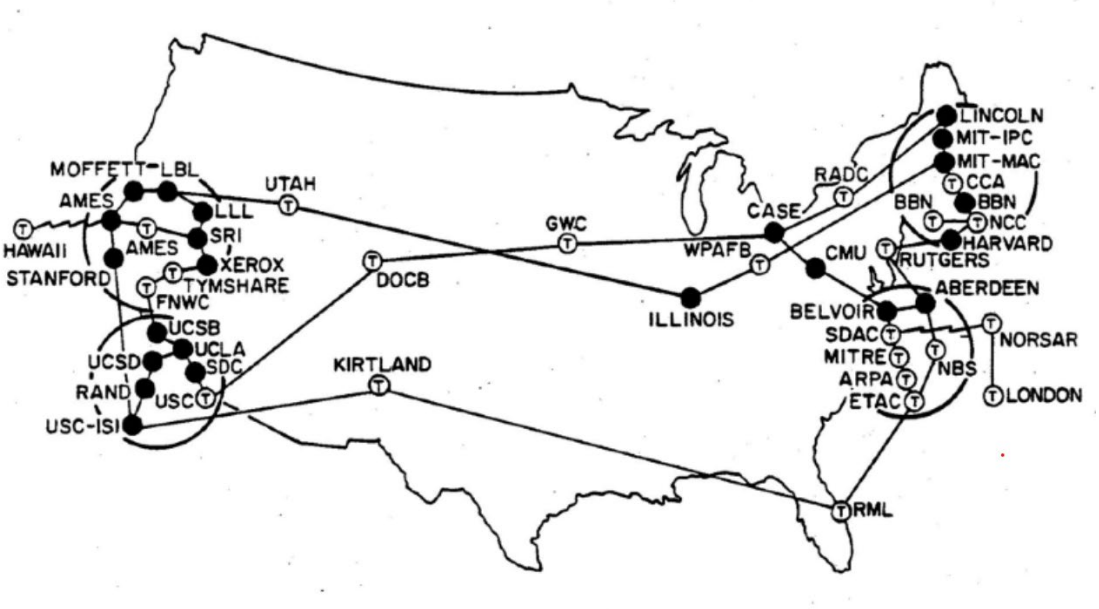

Рисунок 1 — Точки доступу Arpanet

Також цього року було створено перше програмне забезпечення для обміну електронними повідомленнями E-Mail. За короткий час було створено дискусійні форуми, поштові розсилки та інші інструменти віртуальної комунікації[1].

Основна задача чату не змінилась с тих пір і міцно корелює з суттю інтернету — швидкий та безпечний обмін інформацією на будь-якій віддаленості.

У 1980-ті роки чати почали використовуватися для комунікації в рамках локальних мереж, а в 1990-ті роки з'явилися перші публічні чат-сервіси, як-от ICQ(див. рис.2) і AOL Instant Messenger (AIM).

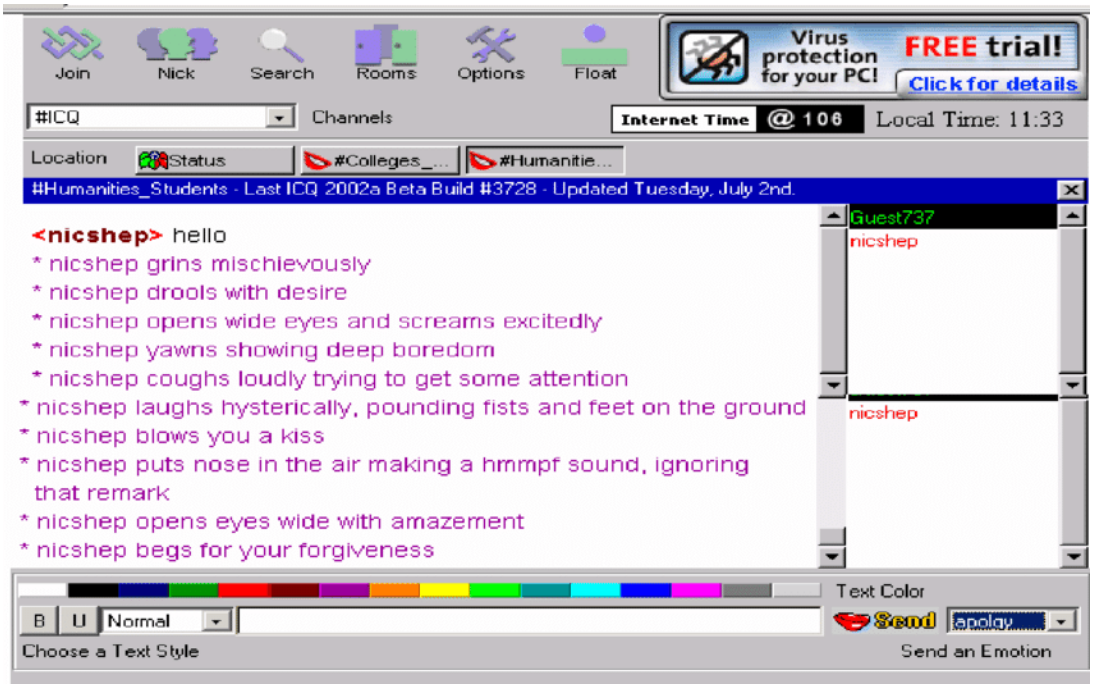

Рисунок 2 — Інтерфейс ICQ

На початку 2000-х років із розвитком інтернету і доступності високошвидкісних з'єднань, з'явилася можливість створення веб-чатів. Одним із перших веб-чатів був створений 2003 року сервіс Meebo.

Сьогодні веб-чати використовуються в різних сферах діяльності і є невід'ємною частиною багатьох онлайн-сервісів. Вони стали більш

функціональними і зручними у використанні, і багато хто з них використовується на мобільних пристроях. З появою нових технологій, таких як штучний інтелект і машинне навчання, веб-чати стали більш автоматизованими і можуть виконувати низку функцій, як-от автоматична відповідь на запитання клієнтів, збір зворотного зв'язку тощо.

Веб-чати продовжують розвиватися і поліпшуватися, додаючи нові функції і можливості. Однією з найпомітніших змін останніх років стала можливість використання веб-чатів як інструменту для бізнесу, особливо для забезпечення зв'язку з клієнтами.

Багато компаній використовують веб-чати для зв'язку з клієнтами, надаючи їм можливість швидко отримати відповіді на свої запитання і проблеми. Веб-чати також використовуються для проведення вебінарів і онлайн-тренінгів, а також для спільної роботи над проектами в режимі реального часу.

З розвитком мобільних пристроїв і технологій, веб-чати стали більш мобільними і доступними, що дає змогу користувачам спілкуватися в будь-який час і з будь-якого місця. Багато веб-чатів також додали можливість використання голосових і відео-дзвінків, що дає змогу користувачам спілкуватися ефективніше й біль особисто.

Сучасні веб-чати також використовуються для соціальної комунікації, особливо в молодіжному середовищі. Вони дають змогу людям спілкуватися з друзями та знайомими в режимі реального часу, обмінюватися фотографіями, відео та іншою інформацією.

Загалом, веб-чати є невід'ємною частиною сучасної комунікаційної інфраструктури, надаючи користувачам швидкий і зручний спосіб спілкування та обміну інформацією в будь-якій точці світу.

На цьому прикладі можна розглянути популярний сучасний месенджер Telegram(див. рис. 3).

— Захист даних і приватність: Telegram використовує протокол шифрування MTProto, який забезпечує високу безпеку даних. Крім того, Telegram дозволяє користувачам встановити двоетапну аутентифікацію.

— Гнучкість: Telegram підтримує не тільки текстові повідомлення, а й файли, фото, відео, голосові повідомлення, стікери, анімовані GIF-зображення та багато іншого.

— Великі групи та канали: Telegram дає змогу створювати групи з понад 200 000 учасників, а також канали з необмеженою кількістю підписників. Це робить додаток корисним для комунікації у великих спільнотах.

— Інтеграція з іншими сервісами.

— Інтерфейс: Telegram має та інтуїтивно зрозумілий інтерфейс. Крім того, у Telegram є функція пошуку, сортування потрібних повідомлень.

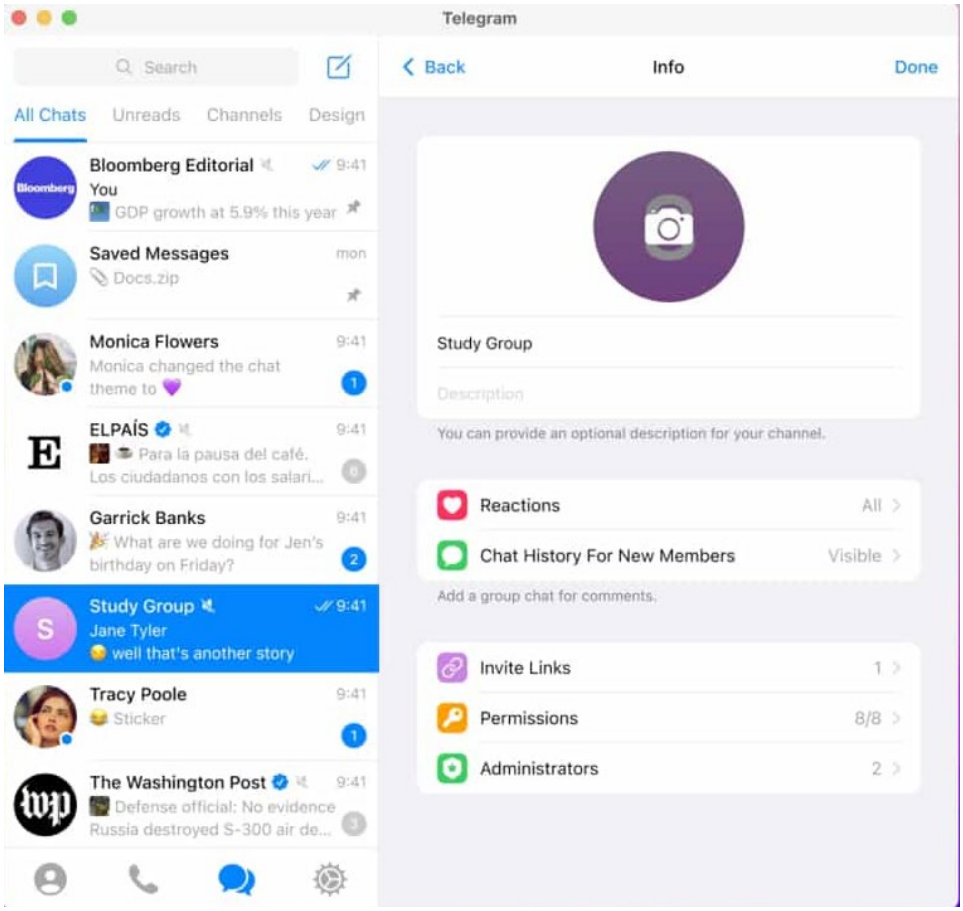

Рисунок 3 — Інтерфейс Телеграму

1.2 Призначення розробки та область застосування.

Темою бакалаврської кваліфікаційної роботи виступає: «Розробка крос платформного додатку для організації чату з використанням NET6, мова C#.». В якості функціональної основи взятий клієнт-серверний підхід. Тобто головною метою роботи є створення одної серверної частини на технології . NET, до якого під’єднано декілька клієнтів різних платформ

Головними критеріями розроблювального є:

— Функціональність.
— Зручність в використанні.
— Стабільність роботи на всіх розроблених клієнтах

Система призначена для:

— Створення та менеджмент групових чатів
— Відправка приватних та групових повідомлень

Система позиціонується як кросплатформний додаток, який дає можливість використовувати клієнти різних платформ для зручної та функціональної комунікації з іншими користувачами.

1.3 Підстава для розробки

Відповідно до ОКХ та ОПП, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу (проект). Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

- ОКХ та ОПП за напрямом підготовки 6.050101 «Комп’ютерні науки»;
- Графік навчального процесу та навчальний план;

- наказ ректора Національного технічного університету «Дніпровська політехніка» № _____ від __.__.2023 р;
- завдання на дипломний проект на тему «Розробка веб додатку для візуалізації планування задач».

1.4 Постановка завдання

Метою дипломного проекта є розробка кросплатформного додатку для візуального планування задач основою якого є клієнт-серверний WEB-API підхід. Система призначена для швидкого та зручного обміну повідомленнями.

Веб-додаток повинен реалізувати такі дії як:

— Збереження даних.
— Надавати можливість використовувати CRUD операції для всіх компонентів системи: повідомлення, користувач, чат.
— Давати змогу інтерактивно взаємодіяти між елементами програмного інтерфейсу.

Для виконання проекту необхідно:

— Проаналізувати існуючі рішення для візуалізації планування задач.
— Спроектувати архітектуру системи.
— Розробити дизайн та зручний інтерфейс додатку.
— Реалізувати front-end та back-end спроектованої системи.

1.5. Вимоги до програми або програмного виробу.

1.5.1. Вимоги до функціональних характеристик.

Розроблене програмне забезпечення, для того, щоб досягнути поставлених цілей, повинно підтримувати виконання таких дій:

- Реагування на дії користувача в онлайн режимі.
- Надання максимально швидкої навігації по додатку.
- Коректна візуалізація елементів веб-додатку.

Для підтримки вище перераховані функцій у додатку має бути реалізовано:

- Підтримка веб-браузера та доступ до програми через нього.
- Програмна та апаратна сумісності.
- Стандартна конфігурація яка дає змогу ввести застосунок в експлуатацію.

1.5.2 Вимоги до інформаційної безпеки.

Для коректної роботи програми потрібно реалізувати:

- Можливість редагування даних.
- Можливість тривалої роботи протягом 24 годин (1 доба).
- Контроль та обробка вхідних даних.
- Збереження цілісності даних у випадку збою системи.

Також система повинна мати наступні характеристики:

- Захист від несанкціонованого доступу.
- Відновлення після збою протягом 10 хвилин.
- Захист даних користувача при повторному підключені.

1.5.3 Вимоги до складу та параметрів технічних засобів.

Для забезпечення надійного функціонування програмного забезпечення необхідно, щоб обчислювальна машина, на якій буде експлуатуватися веб-додаток, мала такі характеристики:

— Маніпулятор “миша”.
— Клавіатура.
— Доступ до онлайн мережі.
— 1 Гб вільного місця на жорсткому диску.
— Процесор Intel Core i3-2348 з тактовою частотою 2.3 ГГц.
— Не менше ніж 4 Гб оперативної пам’яті.
— Рідкокристалічний монітор з діагоналлю 17".

Вище наведені характеристики являють собою рекомендовані. Це означає, що при наявності характеристик не нижче зазначених, розроблений додаток буде функціонувати відповідно до вимог щодо надійності, безпеки та швидкості обробки даних.

1.5.4 Вимоги інформаційної та програмної сумісності.

Для коректного функціонування програми необхідно, щоб програмне забезпечення обчислювальної машини, на якій буде експлуатуватися веб-додаток, відповідало наступним вимогам:

— Операційна система Windows 7/10.
— Мобільний пристрій або емулятор

Серверна сторона має бути реалізована на мові програмування C# з використанням фреймворку .NET, СКБД SQL Server, та ORM Entity Framework Core. Мобільний та комп’ютерний клієнти реалізовані на програмування C# з використанням фреймворку MAUI та мови розмітки XAML

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Під час виконання дипломної роботи було розроблено програму, а саме, кросплатформний додаток для організації чатів, головна мета якого забезпечити зручний та якісний інструментарій для створення користувачів, чатів та відправки повідомлень.

Призначення розробленого додатку:

— Надання можливості створення користувачів, приватних, групових чатів та їх повідомлень.
— Надання можливості керування створеними об’єктами в онлайн середовищі.
— Надання можливості переміщення між сторінками з функціоналом.
— Збереження даних у базі даних.

Для досягнення вище перерахованих функціональних можливостей розроблена програма повинна:

— Мати сумісність зі стандартною апаратною конфігурацією обчислювальної машини.
— Мати підтримку програмного забезпечення, а саме операційних систем Windows 10.
— Мати підтримку сучасних браузерів.

2.2. Опис застосованих математичних методів

Так як особливості предметної області розв'язуваної задачі не передбачають застосування математичних методів, при розробці веб-додатку для візуального планування задач математичні методи не використовувалися.

2.3. Опис використаних технологій та мов програмування

Додаток має бути реалізован на мові програмування C# з використанням фреймворку .NET для Web API, серверної частини, MAUI для Android та desktop клієнтів. Для візуалізації в MAUI було використано XAML який описує зовнішній вигляд сторінки.

Опис мови програмування C#

C# (вимовляється як "See Sharp") - це сучасна об'єктно-орієнтована мова програмування з безпекою типів. C# дозволяє розробникам створювати багато типів безпечних і надійних додатків, які працюють в .NET.

C# - це об'єктно-орієнтована, компонентно-орієнтована мова програмування. C# надає мовні конструкції для безпосередньої підтримки цих концепцій, що робить C# природною мовою для створення та використання програмних компонентів. З моменту свого виникнення C# додала функції для підтримки нових робочих навантажень та нових методів проектування програмного забезпечення. За своєю суттю C# є об'єктно-орієнтованою мовою. Ви визначаєте типи та їхню поведінку.

Декілька функцій C# допомагають створювати надійні та довговічні додатки. Збір сміття автоматично звільняє пам'ять, зайняту недоступними невикористаними об'єктами. Типи, що обнуляються, захищають від змінних, які не посилаються на виділені об'єкти. Обробка винятків забезпечує структурований та розширюваний підхід до виявлення та відновлення помилок. Лямбда-вирази підтримують методи функціонального програмування. Синтаксис Language Integrated Query (LINQ) створює загальний шаблон для роботи з даними з будь-якого джерела. Підтримка мовою асинхронних операцій забезпечує синтаксис для побудови розподілених систем.

C# має уніфіковану систему типів. Всі типи C#, включаючи примітивні типи, такі як int та double, успадковуються від одного кореневого типу об'єкта. Всі типи поділяють набір спільних операцій. Значення будь-якого типу можна зберігати, транспортувати та оперувати з ними у послідовний спосіб. Крім того,

C# підтримує як користувацькі типи посилань, так і типи значень. C# дозволяє динамічно розподіляти об'єкти та зберігати в потоці легкі структури. C# підтримує узагальнені методи та типи, які забезпечують підвищену безпеку типів та продуктивність. C# надає ітератори, які дозволяють реалізаторам класів колекцій визначати власну поведінку для клієнтського коду. [2]

Опис фреймворку .NET та методології

.NET - це платформа з відкритим вихідним кодом для створення десктопних, веб- та мобільних додатків, які можуть працювати на будь-якій операційній системі. Система .NET включає в себе інструменти, бібліотеки та мови, які підтримують сучасну, масштабовану та високопродуктивну розробку програмного забезпечення. Активна спільнота розробників обслуговує та підтримує платформу .NET.

Простіше кажучи, платформа .NET - це програмне забезпечення, яке може виконувати такі завдання:

— Перекладати код мови програмування .NET в інструкції, які може обробити обчислювальний пристрій.
— Надавати утиліти для ефективної розробки програмного забезпечення. Наприклад, визначати поточний час або виводити текст на екран.
— Визначити набір типів даних для зберігання на комп'ютері такої інформації, як текст, числа і дати. [3]

API — набір означень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено - це набір чітко означених методів (функцій) для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення оскільки можна скористуватися готовими об’єктами (функціями) іншого програмного забезпечення через означені в останньому правила взаємодії. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

При використанні прикладного програмного інтерфейсу в контексті веб-розробки, як правило, API означується набором повідомлень-запитів HTTP та структурою повідомлень-відповідей. Повідомлення можуть мати різний формат, як правило цн XML або JSON. Доступ відбувається до з однієї або декількох загальнодоступних кінцевих точок (endpoints).

Кінцеві точки є важливими аспектами взаємодії з веб-інтерфейсами на стороні сервера, оскільки вони вказують, де знаходяться ресурси, доступ до яких може отримати стороння програма. Зазвичай доступ здійснюється через URI, до якого надсилаються HTTP-запити, і звідки очікується відповідь. Кінцеві точки повинні бути статичними, інакше правильне функціонування програмного забезпечення, яке взаємодіє з нею, не може бути гарантоване. Якщо місце розташування ресурсу змінюється (і разом з ним кінцева точка), то раніше написане програмне забезпечення буде перервано, оскільки потрібний ресурс більше не може бути знайдено в одному місці. [4]

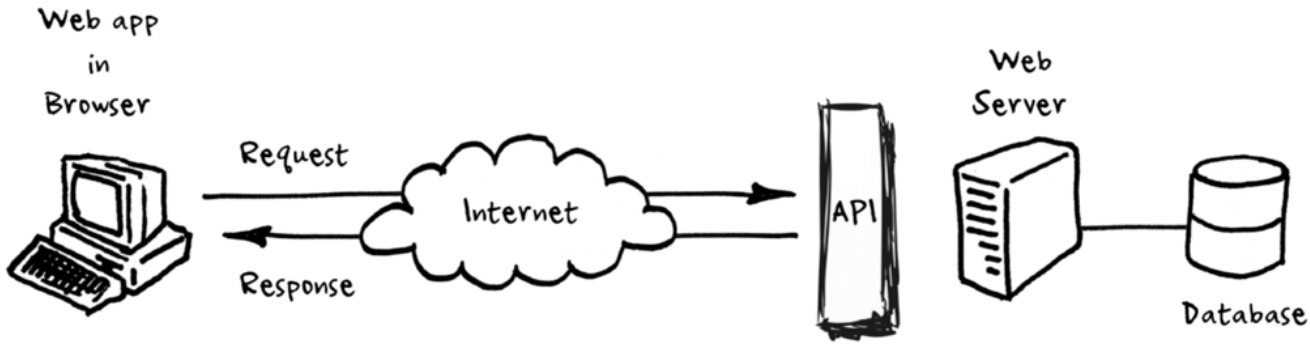

Рисунок 4 — Структура додатку з використанням API

Опис фреймворку MAUI

NET Multi-Platform App UI (.NET MAUI) - це крос-платформна платформа для створення власних мобільних і класичних застосунків за допомогою C# і XAML. За допомогою .NET MAUI можна розробляти додатки, що можуть працювати на Android, iOS, macOS та Windows з однієї спільної бази коду.

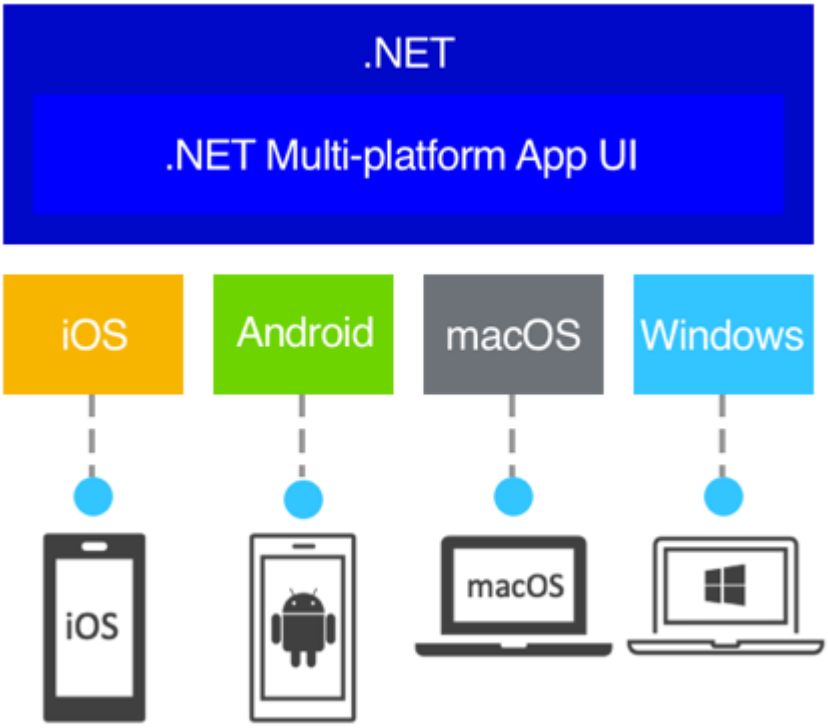

Рисунок 5 — Можливості .NET та MAUI

.NET MAUI - це модель з відкритим вихідним кодом, і це еволюція Xamarin.Forms, розширена від мобільних сценаріїв до класичних сценаріїв, з елементами призначеного для користувача інтерфейсу, які були перебудовані з нуля для підвищення продуктивності та розширюваності. Якщо ви раніше використовували Xamarin.Forms для створення кросплатформних користувацьких інтерфейсів, ви помітите багато схожості з .NET MAUI. Однак між ними також є деякі відмінності. За допомогою .NET MAUI можна створювати багатоплатформні додатки за допомогою одного проєкту, але за потреби можна додати вихідний код і ресурси для конкретної платформи. Однією з основних цілей .NET MAUI є можливість реалізувати в загальній базі коду якомога більше логіки і макета призначеного для користувача інтерфейсу для програми.

.NET MAUI призначений для розробників, які хочуть:

— Створення кросплатформених додатків на XAML і C# з однієї загальної бази коду в Visual Studio.
— Спільне використання макета користувацького інтерфейсу і розробка на різних платформах.
— Спільне використання коду, тестів і бізнес-логіки на різних платформах.[5]

Опис мови XAML

Розширювана мова розмітки додатків (eXtensible Application Markup Language, XAML) - це мова на основі XML, яка є альтернативою програмному коду для створення та ініціалізації об'єктів, а також організації цих об'єктів в ієрархії "батько-дитина".

XAML дозволяє розробникам визначати користувацькі інтерфейси в додатках .NET Multi-platform App UI (.NET MAUI) за допомогою розмітки, а не коду. XAML не є обов'язковим для додатків .NET MAUI, але це рекомендований підхід до розробки інтерфейсу, оскільки він часто є більш лаконічним, більш візуально узгодженим і має підтримку інструментів. XAML також добре підходить для використання з патерном Model-View-ViewModel (MVVM), де XAML визначає подання, яке пов'язане з кодом моделі представлення за допомогою прив'язки даних на основі XAML.

У файлі XAML можна визначати користувацькі інтерфейси, використовуючи всі подання, макети та сторінки .NET MAUI, а також користувацькі класи. Файл XAML можна скомпілювати або вбудувати в пакет програми. У будь-якому випадку, XAML аналізується під час збірки, щоб знайти іменовані об'єкти, а під час виконання об'єкти, представлені в XAML, створюються та ініціалізуються. [6]

Опис СКБД Microsoft SQL Server

Microsoft SQL Server — система управління базами даних, яка розробляється корпорацією Microsoft. Як сервер даних виконує головну функцію по збереженню та наданню даних у відповідь на запити інших застосунків, які можуть виконуватися як на тому ж самому сервері, так і у мережі.

Мова, що використовується для запитів — Transact-SQL, створена спільно Microsoft та Sybase. Transact-SQL є реалізацією стандарту ANSI / ISO щодо структурованої мови запитів SQL із розширеннями. Використовується як для невеликих і середніх за розміром баз даних, так і для великих баз даних масштабу підприємства. Багато років вдало конкурує з іншими системами керування базами даних.

Microsoft та інші компанії пропонують велику кількість програмних засобів розробки, які дозволяють розробляти застосунки для бізнесу з використанням баз даних Microsoft SQL Server. Microsoft SQL Server 2005 включає також Common Language Runtime (CLR) Microsoft .NET, що дозволяє застосункам, розробленим на мовах платформи .NET (наприклад, VB.NET або C#), реалізовувати процедури, що зберігаються та різні функції. Попередні версії засобів розробки Microsoft використовували лише API для надання функціонального доступу до Microsoft SQL Server. [7]

Опис ORM Entity Framework Core

Entity Framework (EF) Core - це легка, розширювана, відкрита та крос-платформна версія популярної технології доступу до даних Entity Framework. EF Core може служити в якості об'єктно-реляційного маппера (O/RM), який:

— Дозволяє .NET розробникам працювати з базою даних, використовуючи об'єкти .NET.
— Усуває необхідність у написанні більшої частини коду доступу до даних, який зазвичай потрібно писати.

— EF Core підтримує багато механізмів баз даних, докладніше див. у розділі Провайдери баз даних.

В EF Core доступ до даних здійснюється за допомогою моделі. Модель складається з класів сутностей та об'єкта контексту, який представляє сеанс роботи з базою даних. Контекстний об'єкт дозволяє запитувати і зберігати дані. Для отримання додаткової інформації див. розділ Створення моделі.

EF підтримує наступні підходи до розробки моделей:

— Згенерувати модель з існуючої бази даних.
— Вручну закодувати модель відповідно до бази даних.

Після створення моделі використовуйте EF Migrations для створення бази даних на основі моделі. Міграції дозволяють розвивати базу даних по мірі того, як змінюється модель. [8]

2.4. Опис структури системи та алгоритмів її функціонування

Для серверної частини мого проекту було обрано стиль багаторівневої архітектури, також відомої як чистоі. У рамках чистої архітектури центральним елементом застосунку є його бізнес-логіка та модель. У цьому випадку бізнес-логіка не залежить від доступу до даних або інших інфраструктур, тобто стандартна залежність інвертується: інфраструктура та деталі реалізації залежать від ядра додатка. Ця функціональність досягається шляхом визначення абстракцій або інтерфейсів у ядрі додатка, які реалізуються типами, визначеними в шарі інфраструктури. Таку архітектуру зазвичай малюють у вигляді серії кіл зі спільним центром, яка зовні нагадує зріз цибулини. На рис. 6 показано приклад такого стилю представлення архітектури.[9]

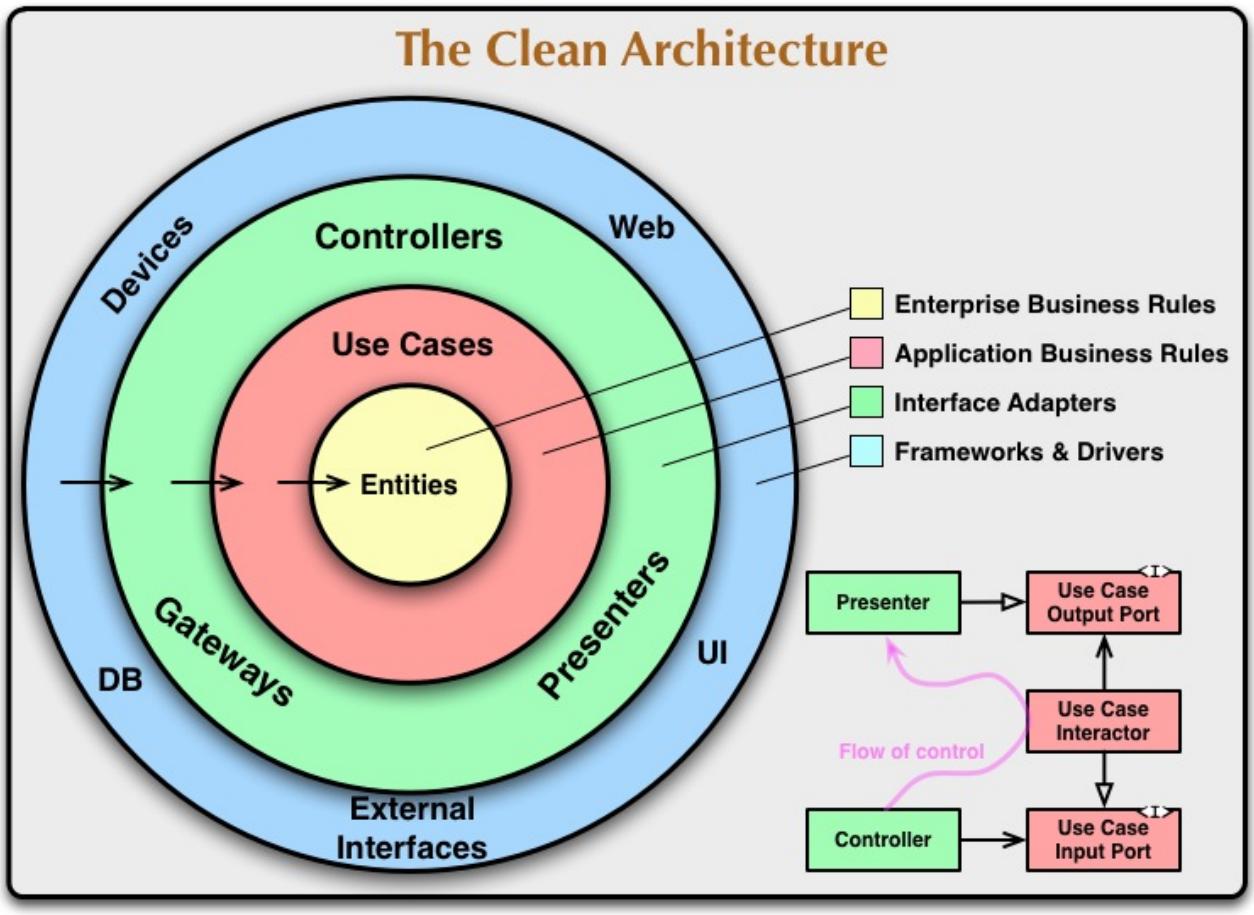

Рисунок 6 — Шарове представлення архітектури

На цій схемі залежності спрямовані з самого внутрішнього кола. Ядро додатка називається так тому, що знаходиться в самому центрі цієї схеми. Як видно на схемі, ядро додатка не має залежностей від інших шарів додатка. Сутності та інтерфейси додатка знаходяться в самому центрі. Одразу після них, але все ще в межах ядра програми, розташовані доменні служби, які зазвичай реалізують інтерфейси, визначені у внутрішньому колі. За межами ядра програми розташовуються шари користувацького інтерфейсу та інфраструктури, які залежать від ядра програми, але не один від одного (обов'язково).

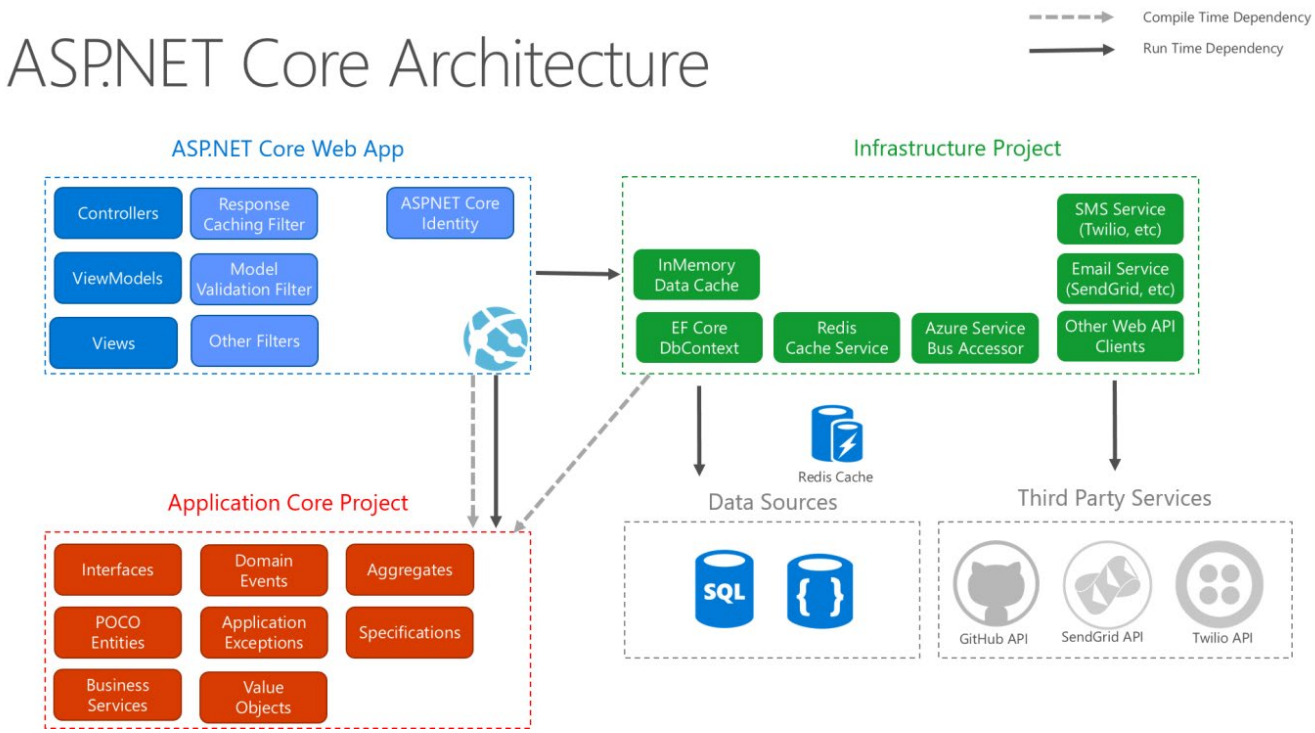

Рисунок 7 — Схема чистої архітектури .NET

Ось розбивка компонентів та їх зв'язків:

Моделі: Це класи, які представляють сутності даних у додатку. Приклади включають User, GroupChat і GroupMessage(див. лістинг 1).

Лістинг 1 — Модель даних користувача:

```
namespace KoalitionServer.Models
{
  public class User
  {
    public int UserId { get; set; }
    public string Login { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
    public bool? Online { get; set; }
    public DateTime? LastRescent { get; set; }

    //one user to many messages and groups
    public ICollection<PrivateMessage> PrivateMessages { get; set; }
    public ICollection<GroupMessage> GroupMessages { get; set; }
    public ICollection<GroupChatsToUsers> GroupChatsToUsers { get; set; }
    public ICollection<PrivateChat> PrivateChats { get; set; }
    public ICollection<PrivateChatsToUsers> PrivateChatsToUsers { get; set; }

  }
}
```

Контролери: Контролери обробляють вхідні HTTP-запити і надають відповідні відповіді. Наприклад, автентифікація користувачів, керування груповим чатом та обробка приватних чатів, створення користувачів(див. лістинг 2).

Лістинг 2 — Контролер з ендпоінтом для виведення існуючих користувачів:

```
namespace KoalitionServer.Controllers
{
  [ApiController]
  [Route("api/[controller]")]
  public class UsersController : ControllerBase
  {
    private readonly IMediator _mediator;
    private readonly UserService _userService;
    private readonly AppDbContext _context;
    private readonly IHttpContextAccessor _httpContextAccessor;

    public UsersController(UserService userService, AppDbContext appDbContext, IMediator mediator,
IHttpContextAccessor httpContextAccessor)
    {
      _userService = userService;
      _context = appDbContext;
      _mediator = mediator;
      _httpContextAccessor = httpContextAccessor;
    }

    [HttpPost("register")]
    public async Task<ActionResult<User>> RegisterUser([FromBody] RegistrationRequest regRequest)
    {
      var user = await _userService.RegisterUser(regRequest);
      return Ok(user);
    }
```

Сервіси: Сервіси інкапсулюють бізнес-логіку додатку і надають функціональність для маніпулювання даними та виконання операцій. У моєму випадку є такі сервіси, як UserService(див. лістинг 3), CreateGroupChatService, GetCurrentUserGroupChatsService, AddUserToGroupChatService тощо. Ці сервіси взаємодіють з базою даних і виконують операції на основі запитуваної функціональності.

Лістинг 3 — Логіка сервісу, що необхідний для створення користувача:

```
namespace KoalitionServer.Services.UserServices
{
  public class UserService
  {
    private readonly AppDbContext _context;
```

```
    private readonly IPasswordHasher<User> _passwordHasher;
    private readonly IConfiguration _configuration;
    private readonly IHttpContextAccessor _httpContextAccessor;

    public UserService(AppDbContext context, IPasswordHasher<User> passwordHasher, IConfiguration configuration,
IHttpContextAccessor httpContextAccessor)
    {
        _context = context;
        _passwordHasher = passwordHasher;
        _configuration = configuration;
        _httpContextAccessor = httpContextAccessor;
    }

    public async Task<User> RegisterUser(RegistrationRequest regRequest)
    {
        if (await _context.Users.AnyAsync(u => u.Email == regRequest.Email))
        {
            throw new ArgumentException("User with this email already exist!");
        }

        var newUser = new User
        {
            Login = regRequest.Login,
            Name = regRequest.Name,
            Email = regRequest.Email,
            Password = _passwordHasher.HashPassword(null, regRequest.Password)
        };

        _context.Users.Add(newUser);
        await _context.SaveChangesAsync();

        return newUser;
    }
```

Клієнтська частина була зроблена за архітектурним шаблоном MVVM. Шаблон MVVM допомагає чітко відокремити бізнес-логіку застосунку та логіку представлення від користувацького інтерфейсу. Чіткий поділ між логікою застосунку та користувацьким інтерфейсом допомагає розв'язати численні проблеми розроблення та спрощує тестування, обслуговування та розвиток застосунку. Це також може значно поліпшити можливості повторного використання коду і дає змогу розробникам і дизайнерам користувацького інтерфейсу легше взаємодіяти під час розроблення відповідних частин програми.

Шаблон MVVM містить три основні компоненти: модель, подання та модель подання. Кожен із них служить певній меті. На схемі нижче показано зв'язки між трьома компонентами.[10]

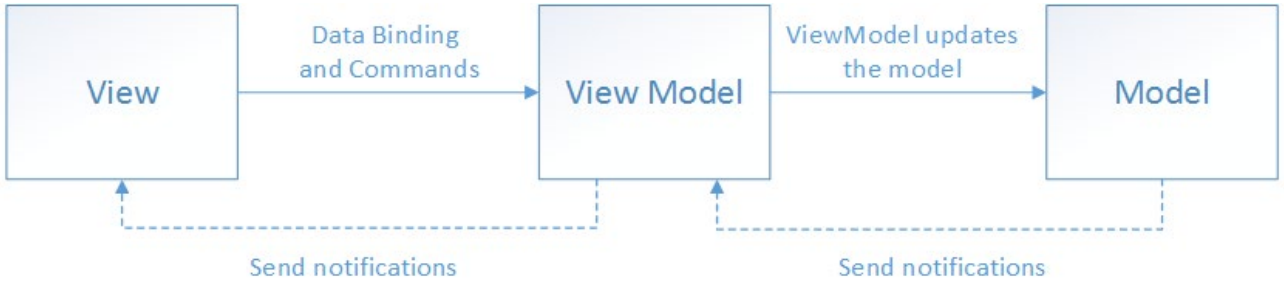

Рисунок 8 — Схема зв’язку компонентів MVVM

Ось розбивка компонентів та їх зв’язків:

Models: Моделі в клієнті MAUI представляють сутності даних або об'єкти передачі даних (DTO), що використовуються в клієнтській програмі. Ці моделі відповідають за зберігання та представлення даних, які отримуються з бекенд-сервісів або надсилаються до них(див. лістинг 4).

Лістинг 4 — Модель даних запиту на створення групового чату:

```
namespace KoalitionAndroidClient.Models
{
  public class CreateGroupChatRequest
  {
    public int Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
  }
}
```

Views: Подання у клієнті MAUI - це візуальні елементи, з якими взаємодіє користувач. Вони відповідають за відображення користувацького інтерфейсу і збір даних, введених користувачем. Прикладами представлень є екрани входу, чату та створення групового чату(див. лістинг 5).

Лістинг 5 — Інтерфейс створення чату

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="KoalitionAndroidClient.Views.AddGroupChatPage.AddGroupChatPage"
        Title="AddGroupChatPage">
  <VerticalStackLayout
    Spacing="25"
      Padding="30,0"
      VerticalOptions="Center">

    <Label
        Text="Create your chat"
        SemanticProperties.HeadingLevel="Level1"
        FontSize="32"
        HorizontalOptions="Center" />
```

```
    <Frame HeightRequest="45" Margin="-20,0,0,0" Padding="0" HasShadow="True" BorderColor="White"
HorizontalOptions="FillAndExpand">
        <Entry Text="{Binding Name}" Margin="20,0,0,0" VerticalOptions="Center" Placeholder="Name" />
    </Frame>

    <Frame HeightRequest="45" Margin="-20,0,0,0" Padding="0" HasShadow="True" BorderColor="White"
HorizontalOptions="FillAndExpand">
        <Entry Text="{Binding Description}" Margin="20,0,0,0" VerticalOptions="Center" Placeholder="Desciption"
IsPassword="True" />
    </Frame>

    <Button Text="Done" WidthRequest="100" CornerRadius="20" HorizontalOptions="Center" Command="{Binding
CreateGroupChatCommand}" />
  </VerticalStackLayout>
</ContentPage>
```

ViewModels: Модель-подання у клієнті MAUI виступають посередником між поданнями та внутрішніми сервісами. Вони містять логіку представлення та логіку маніпулювання даними, необхідну для подання. ViewModels відповідають за отримання даних з бекенд-сервісів, оновлення представлень та обробку взаємодії з користувачем. Вони розкривають властивості та команди, до яких прив'язуються подання для відображення та маніпулювання даними. Наприклад, команда створення групового чату(див. лістинг 6), отримання групових чатів користувача тощо.

Лістинг 6 — Команда на клієнті, що дозволяє створити новий груповий чат :

```
public ICommand CreateGroupChatCommand { get; set; }
    public AddGroupChatPageViewModel()
    {
        CreateGroupChatCommand = new Command<object>(async (param) => await CreateGroupChat());
    }

    public async Task CreateGroupChat()
    {
        using var client = new HttpClient();
        client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", App.Token);
        var groupChat = new CreateGroupChatRequest
        {
            Name = Name,
            Description = Description
        };

        var groupChatJson = JsonConvert.SerializeObject(groupChat);
        var content = new StringContent(groupChatJson, Encoding.UTF8, "application/json");
        var response = await
client.PostAsync($"{ApiPlatformUrlHelper.GetPlatformApiUrl()}/api/GroupChat/createGroupChat", content);

        await Shell.Current.GoToAsync("///MenuPage");
        if (Application.Current.MainPage is MenuPage menuPage && menuPage.BindingContext is MenuPageViewModel
menuViewModel)
        {
```

```
            menuViewModel.GetGroupChats();
            await Application.Current.MainPage.Navigation.PushAsync(menuPage);
        }
        else
        {
            Console.WriteLine("Error: Unable to cast MainPage to MenuPage or BindingContext to MenuPageViewModel.");
        }
    }
```

Загалом, патерн MVVM використовується для відокремлення проблем представлення даних і бізнес-логіки від інтерфейсу користувача. ViewModels відповідають за пошук даних, маніпуляції з ними та логіку взаємодії, в той час як Views зосереджуються на відображенні елементів інтерфейсу та прив'язці до властивостей ViewModel. Сервіси відповідають за зв'язок з внутрішніми сервісами, абстрагуючись від деталей ViewModels.

Загальний хід роботи програми можна підсумувати наступним чином:

1. Клієнт MAUI надсилає HTTP-запити до відповідних кінцевих точок, наданих внутрішніми контролерами.
2. Контролери отримують запити і викликають відповідні методи сервісів.
3. Сервіси взаємодіють з рівнем даних (базою даних) і виконують необхідні операції.
4. Сервіси повертають результати контролерам.
5. Контролери генерують відповідні відповіді та надсилають їх назад клієнту MAUI.

Така архітектура дозволяє розділити обов'язки і сприяє модульності та ремонтопридатності додатку. Клієнт MAUI виступає як рівень користувацького інтерфейсу, в той час як бекенд-сервіси відповідають за бізнес-логіку та доступ до даних.

Файлова структура серверної частини виглядає так(див. рис 9):

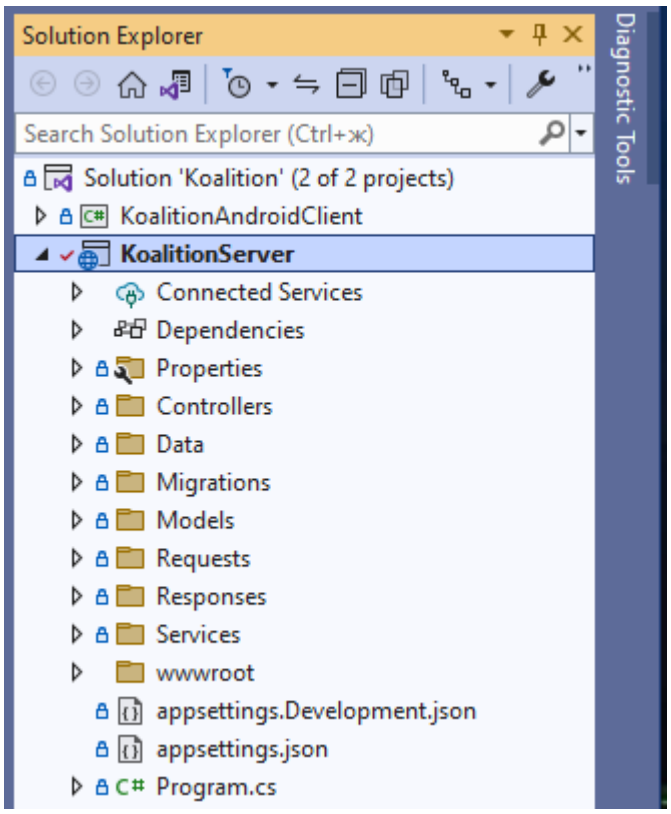

Рисунок 9 — Файлова структура серверної частини

— Connected Services — підключення до SQL Server

— Dependencies — під'єднані бібліотеки у вигляді nuget-пакетів

— Properties — параметри запуску додатку

— Controllers — контролери з HTTP-методами

— Data — контекст EF Core для створення таблиць бази даних

— Migrations — зміни моделей сутностей або контексту, зафіксовані у вигляді міграцій

— Models — моделі сутностей

— Requests — моделі запитів до сервісів

— Responses — моделі відповідей сервісів

— Services — інкапсульована логіка HTTP-методів

— wwwroot — коренева веб-директорія проекту

Розглянемо компонент Dependencies(див. рис. 10): він включає Mediatr — для гнучкого використання моделей відповіді/запиту для сервісів, JwtBearer — для реалізації авторизації, EF Core — ORM для створення таблиць по

моделям, SwaggerUI — інтерфейс розробника для документації та перевірки API

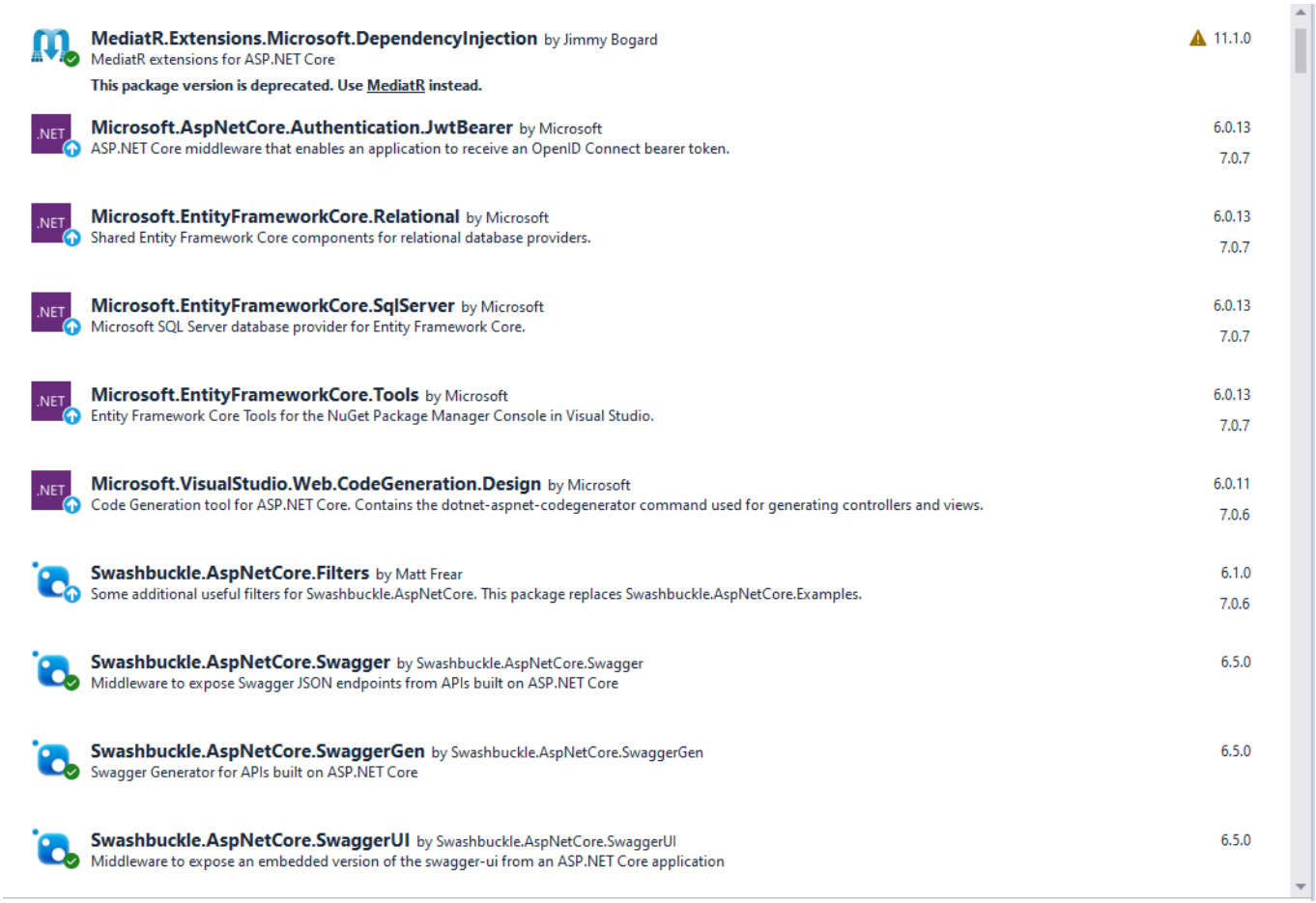

Рисунок 10 — Меню з інстальованими nuget-пакетами

Розглянемо компонент Controllers(див. рис. 11): він включає 4 контролери:

— GroupChat — включає в себе Http-методи для створення, видалення, зміни групового чату, додавання та видалення користувача, відображення чатів
— GroupMessages — включає в себе Http-методи для відправки, видалення, зміни, відображення групових повідомлень
— PrivateChats — включає в себе Http-методи для відправки та створення приватного чату, видалення, зміни, відображення приватних повідомлень
— Users — включає в себе Http-методи для відображення, видалення, зміни, створення, авторизації користувачів, виведення інформації авторизованого користувача

Рисунок 11 — Контролери у інтерфейсі Сваггеру мого серверного додатку

Розглянемо компонент Models, Requests, Responses(див. рис. 12). Ці модулі включають в себе різноманітні структури даних, для створення таблиць та взаємодії з контролерами. Кожна модель — таблиця в БД.

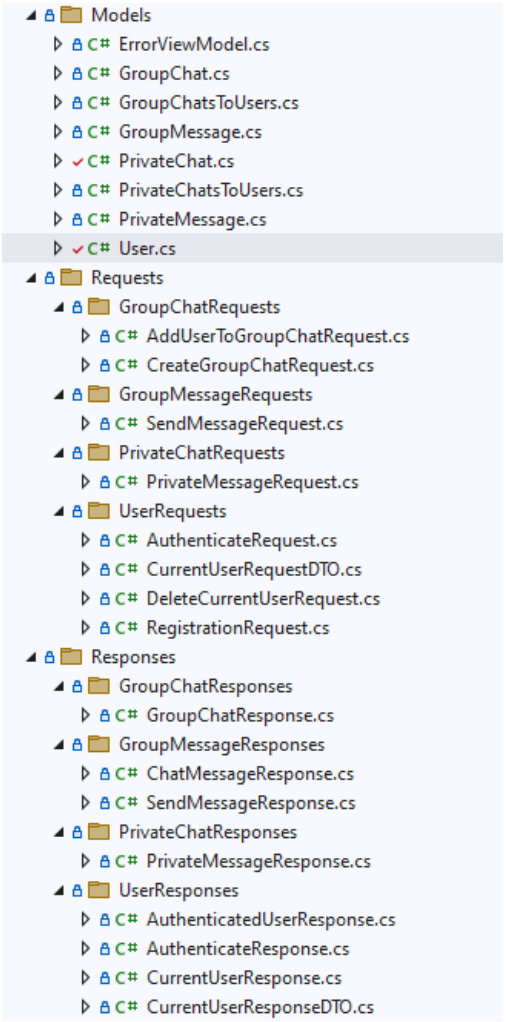

Рисунок 12 — Наявні структури даних

Розглянемо компонент Services(див. рис. 13). В сервісах інкапсульована логіка всіх HTTP-методів для зручного написання та підтримки. Вони розбиті за сутностями: UserServices — операції з користувачем, GroupChatServices — операції з груповим чатом, GroupMessageServices — операції з груповим повідомленням, PrivateChatService — операції з приватними повідомленнями

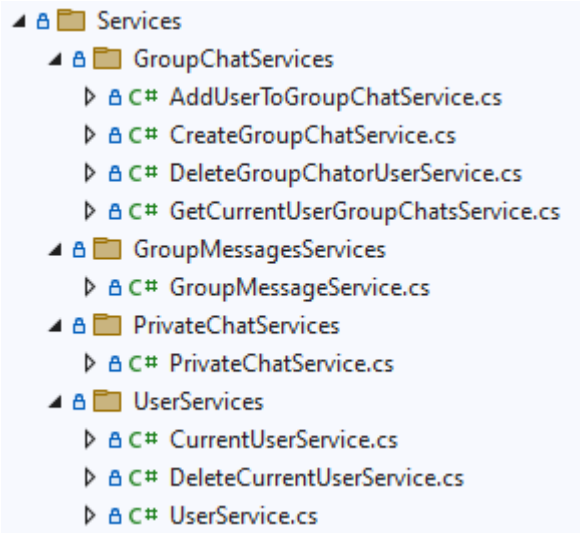

Рисунок 13 — Наявні сервіси

Файлова структура клієнтської частини виглядає так:

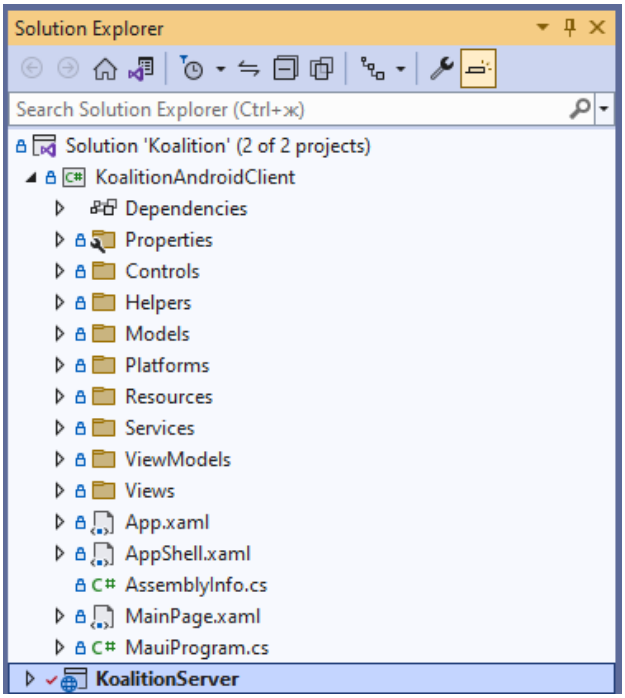

Рисунок 14 — Файлова структура клієнтського проекту

— Dependencies — під’єднані бібліотеки у вигляді nuget-пакетів

— Properties — параметри запуску додатку

— Models — моделі сутностей, запитів та відповідей сервісів

— Services — інкапсульована логіка викликання HTTP-методів

— Controls — реалізація бокового вікна

— Helpers — допоміжний метод для звертання до контролерів при різних типах запуску клієнту

— Resources — всі необхідні інтерфейсні елементи

— ViewModels — файли з логікою викликів API для View

— View — сторінки інтерфейсу

Розглянемо компонент Dependencies(див. рис. 15): він включає Newtonsoft.Json — бібліотеку для роботи з форматом даних JSON, Microsoft.Toolkit.Mvvm — бібліотеку з інструментами реалізації шаблону MVVM, .NET MAUI NuGet dependencies pack — це набір пакетів NuGet, спеціально розроблених для розробки багатоплатформних додатків .NET App UI (MAUI). Ці пакунки надають різні функціональні можливості та бібліотеки, необхідні для створення MAUI-додатків.

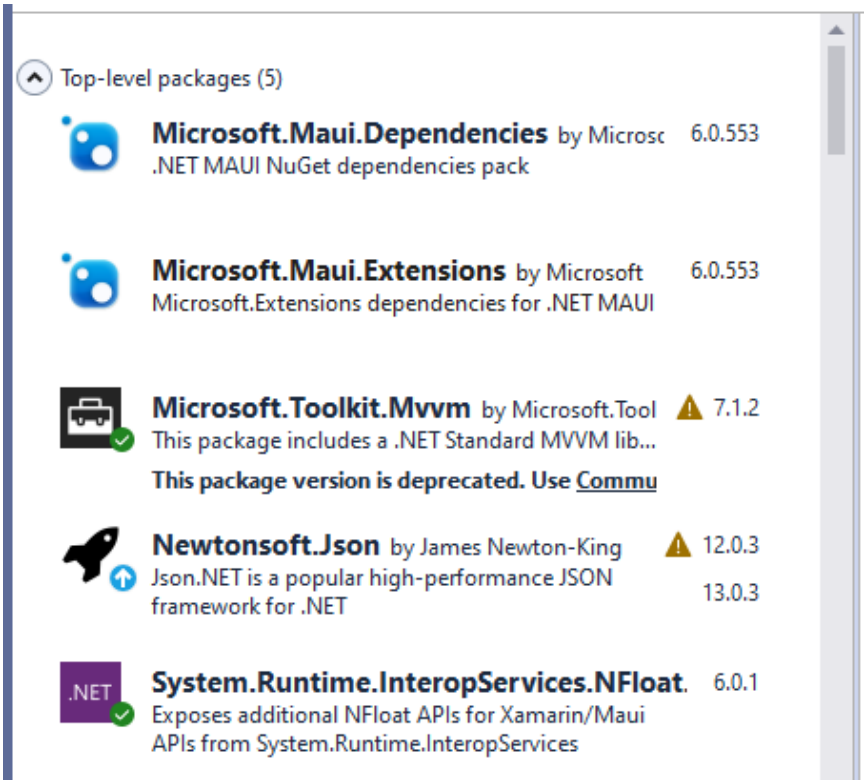

Рисунок 15 — Під’єднані бібліотеки клієнту

Розглянемо компонент Heplers: він містить один допоміжний метод для перемикання посилань на HTTP-методи в залежності від платформи, в режимі якого запущено додаток(див. лістинг 7):

Лістинг 7 — Допоміжний метод GetPlatformApiUrl:

```
namespace KoalitionAndroidClient.Helpers
{
   public static class ApiPlatformUrlHelper
   {
      public static string GetPlatformApiUrl()
      {
#if __ANDROID__
      return "http://10.0.2.2:5127";
#else
      return "http://localhost:5127";
#endif
      }
   }
}
```

Розглянемо компонент Models (див. рис. 16). Ці модулі включають в себе різноманітні структури даних для взаємодії з командами та контролерами

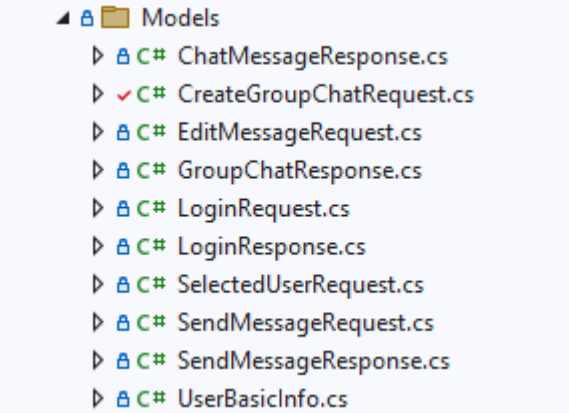

Рисунок 16 — Компонент Models

Розглянемо компонент ViewModels(див. рис. 17): він включає наступні файли з логікою викликів API для View:

— AddGroupChatPageViewModel — реалізація команди для створення групового чату

— GroupChatPageViewModel — реалізація команд для відправки, зміни, та відображення повідомлень

— PrivateChatPageViewModel — реалізація команд для створення приватного чату при відправці першого повідомлення, відправка та відображення повідомлення

— MenuPageViewModel — реалізація команд переходу до групового та приватного чатів, переходу до сторінки створення приватного чату, відображення приватних та групових чатів, користувачів

— LoginPageViewModel — реалізація авторизації та реєстрації, навігації до сторінки Меню

— AppShellViewModel — реалізація логіки бокового вікна: відображення даних користувача та повернення до сторінки авторизації

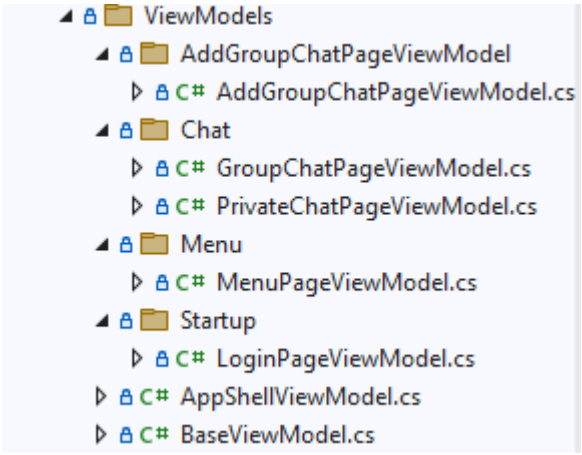

Рисунок 17 — Компонент ViewModels

Система розділена різні модулі, що дозволяє легко орієнтуватися та розуміти її суть, легко вносити зміни та розширювати її. Функціонал логічно структуровано, що дозволяє зручно орієнтуватись в системі та підтримувати її. Точкою входу в веб-додаток є файл Koalition.sln, що знаходиться в папці Koalition.

2.5. Обгрунтування та організація вхідних та вихідних даних програми

Додаток отримує вхідні данні с заповнених користувачем форм веб-компонентів, а також з місця розташування даних компонентів.

Вихідні данні представлені в виді:

— Сторінок додатку.

— Інтерактивних компонентів системи.

2.6. Опис роботи розробленої системи

2.6.1. Використані технічні засоби

Так як вся інформація зберігається на стороні клієнта локально, то наступні рекомендовані характеристики обчислювальної машини наведені виключно для клієнтського обладнання:

- Процесор Intel(R) Core(TM) i3-2348M з тактовою частотою 2.3GHz.
- Оперативна пам’ять в розмірі 4ГБ пам’яті.
- Вільного місця на диску в розмірі 1ГБ пам’яті.
- Рідкокристалічний монітор з діагоналлю не менше 17 ".
- Маніпулятор «миша».
- Клавіатура.
- Доступ до глобальної мережі.

Вище наведені характеристики являють собою рекомендовані. Це означає, що при наявності характеристик не нижче зазначених, розроблений додаток буде функціонувати відповідно до вимог щодо надійності, безпеки та швидкості обробки даних.

2.6.2. Використані програмні засоби

Додаток реалізований за допомогою фреймворків .NET та MAUI. Ці технології можуть працювати на Windows, Android. Для візуалізації компонентів був використаний XAML.

Необхідні програмні засоби на стороні клієнта:

— Операційна система Windows /Android

2.6.3. Виклик та завантаження програми

Розроблений додаток працює на Windows та Android. Його можна запустити файлами KoalitionServer.exe та KoalitionAndroidClient.exe.

2.6.4. Опис інтерфейсу користувача

Для того щоб почати роботу з додатком потрібно запустити серверну так клієнтську частини через exe-файли або IDE. Після цього користувача буде направлено до сторінки авторизації:

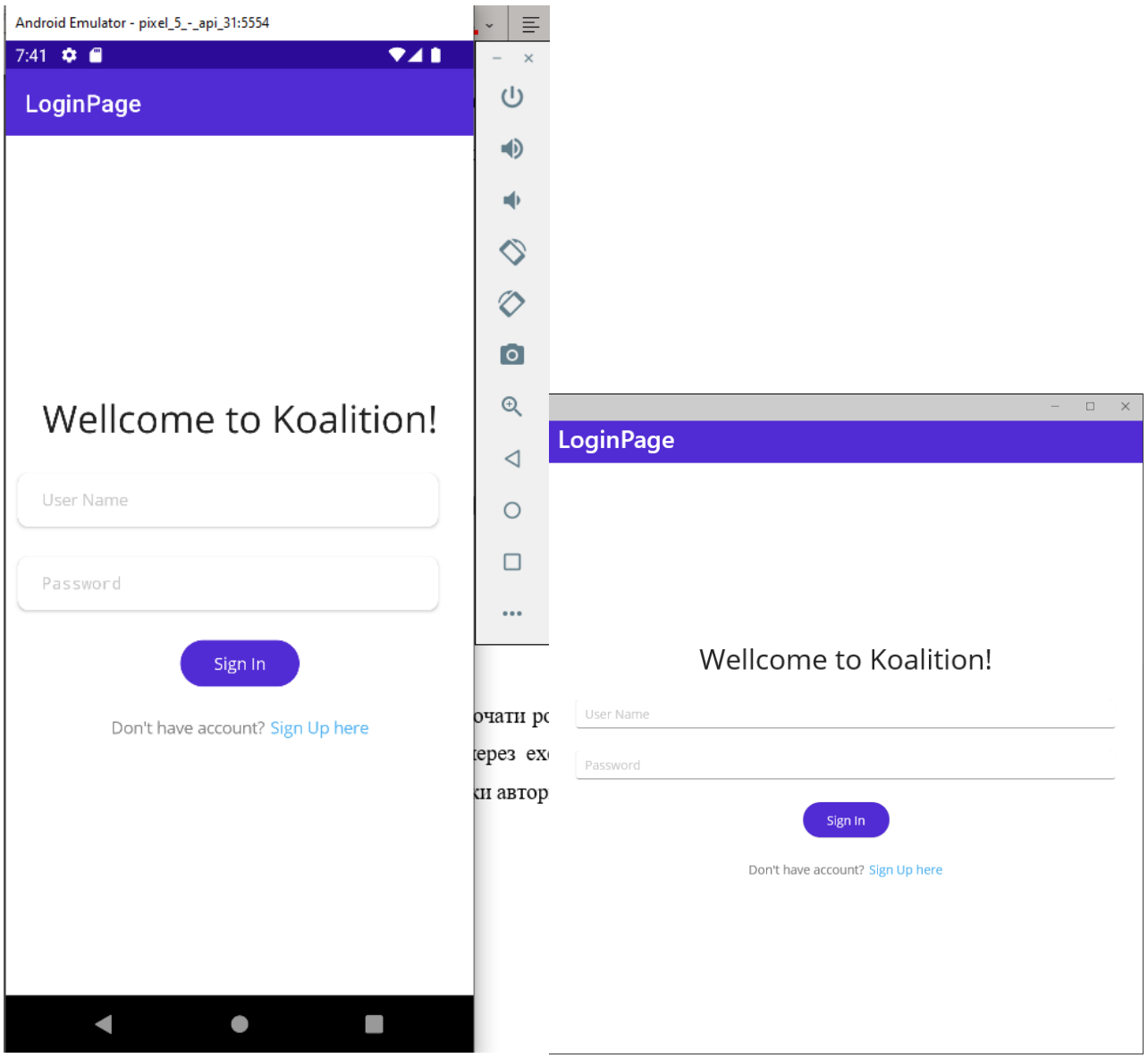

Рисунок 18 — Сторінка авторизації на Android, рисунок 19 — Сторінка авторизації на Windows

При введенні даних та натискання кнопки відбувається авторизація та переміщення до сторінки Меню, на якій відображаються всі існуючі користувачі та чати, в яких приймає участь авторизований користувач:

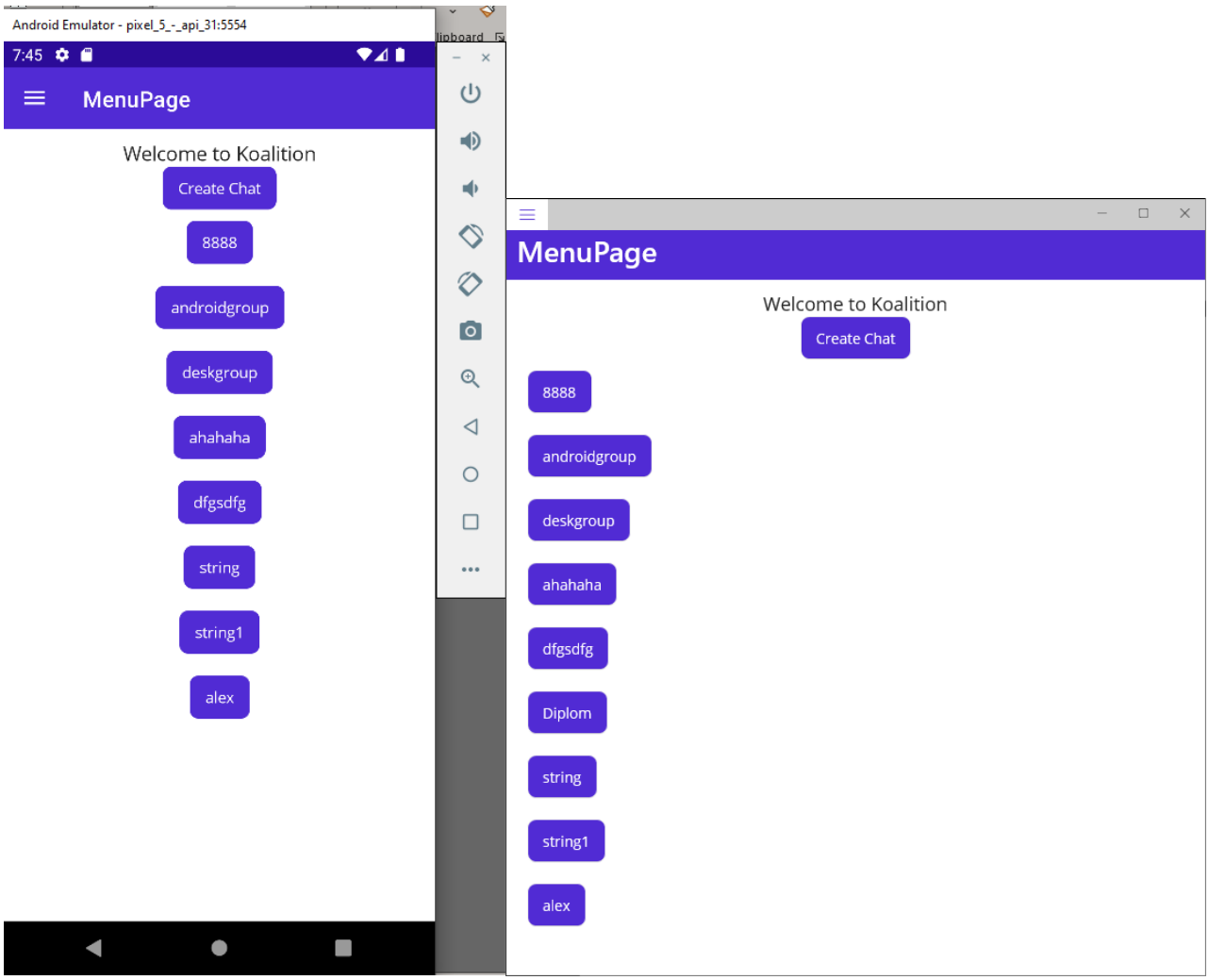

Рисунок 20 — Сторінка меню на Android, рисунок 21 — Сторінка меню на Windows

Якщо користувач бажає створити новий груповий чат, то необхідно натиснути на кнопку Create Chat(див. рис. 22, 23), після чого заповнити поля чату з назвою та описом. Після створення чату відбувається автоматична навігація до головного меню, де вже можна перейти до створеного чату, для цього треба клікнути на нього

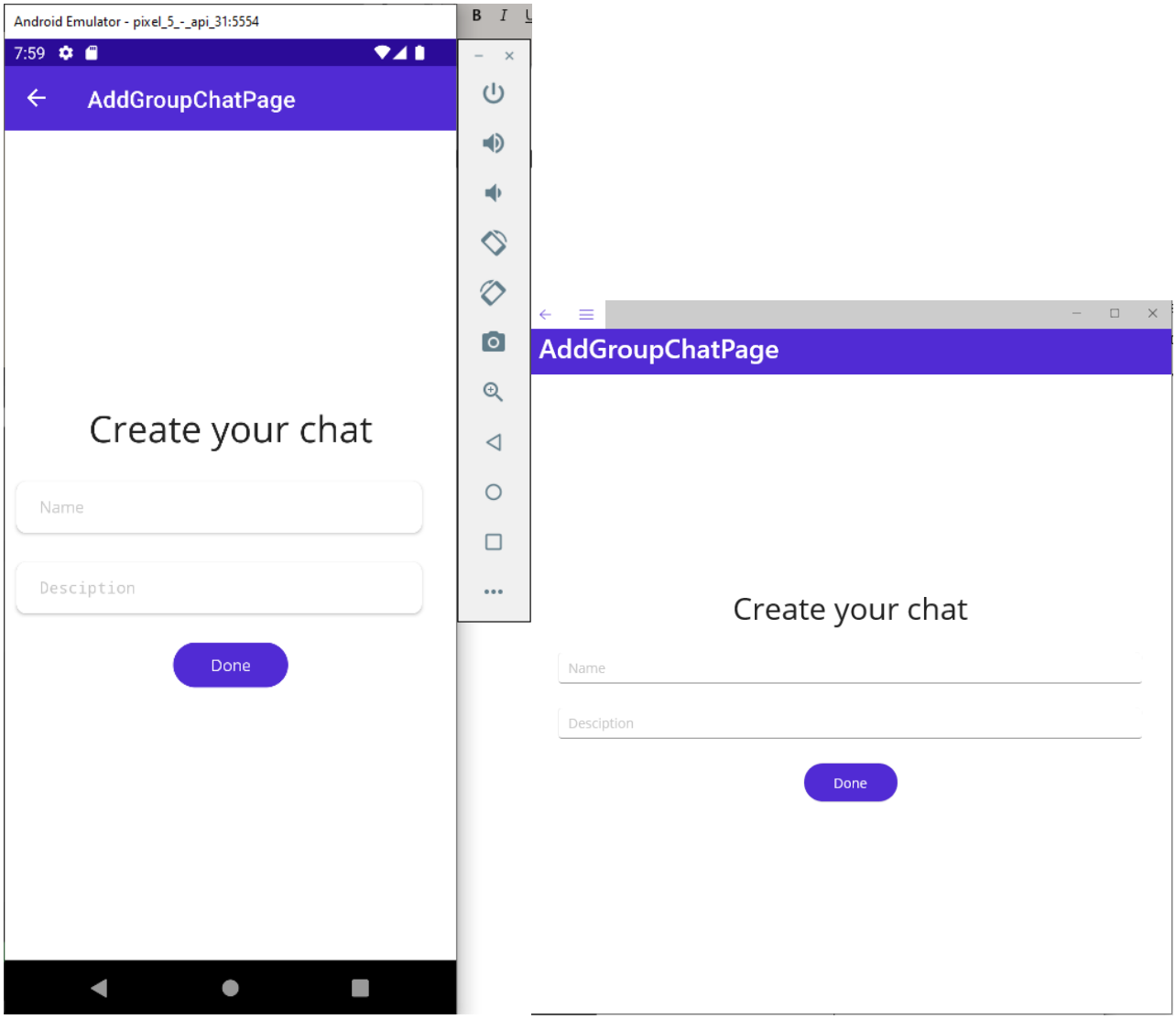

Рисунок 22 — Сторінка створення чату на Android, рисунок 23 — Сторінка створення чату на Windows

Для відправки повідомлення у груповому чаті потрібно лише ввести його та натиснути кнопку Send:

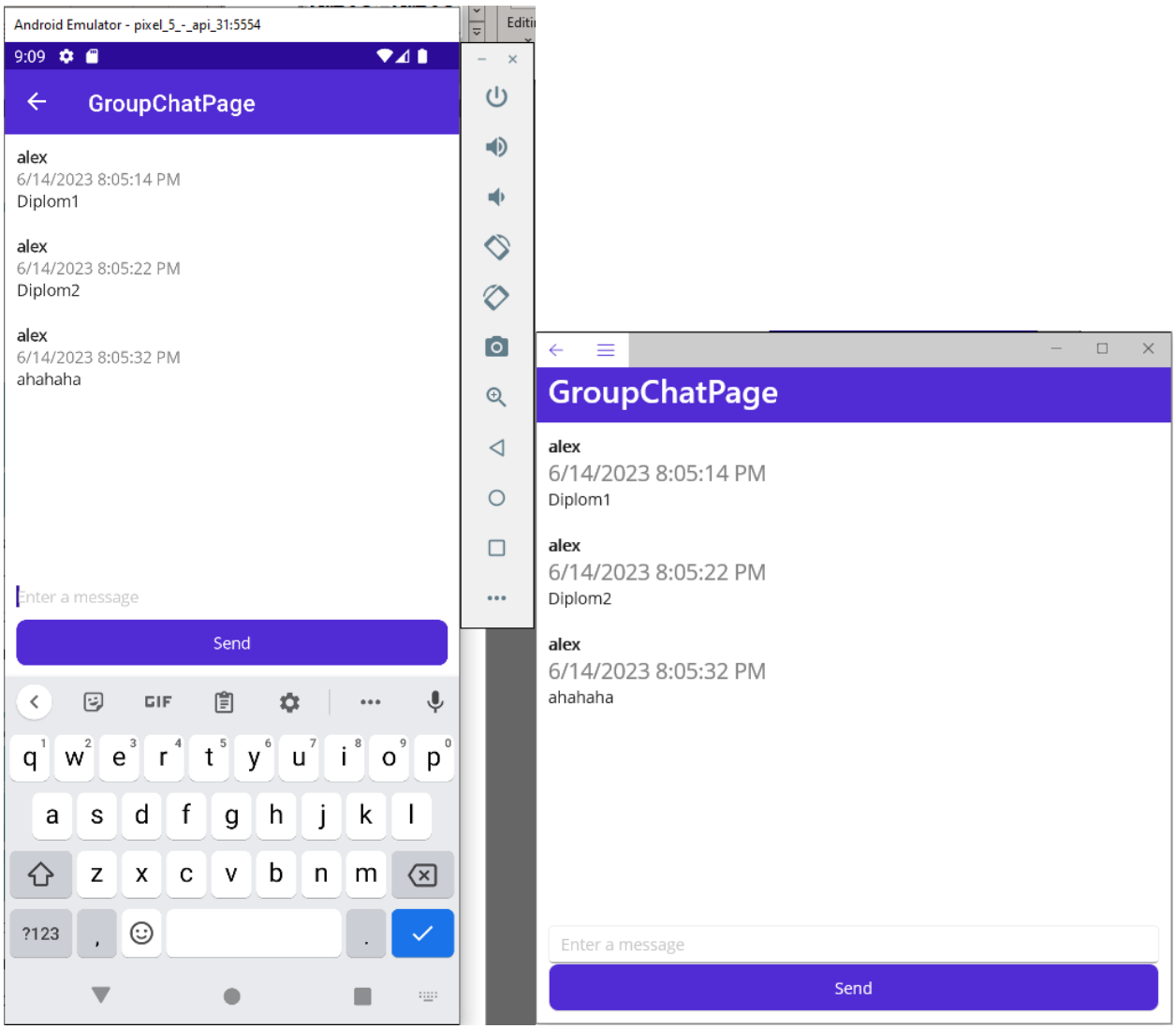

Рисунок 24 — Сторінка групового чату на Android, рисунок 25 — Сторінка групового чату на Windows

Для відправки приватних повідомлень потрібно натиснути на існуючого користувача у Меню, відбудеться перехід до сторінки чату, ввести повідомлення та натиснути Send. Якщо це перше повідомлення користувачу від вас, то автоматично у базі даних створиться приватний чат з вами при відправці повідомлення:

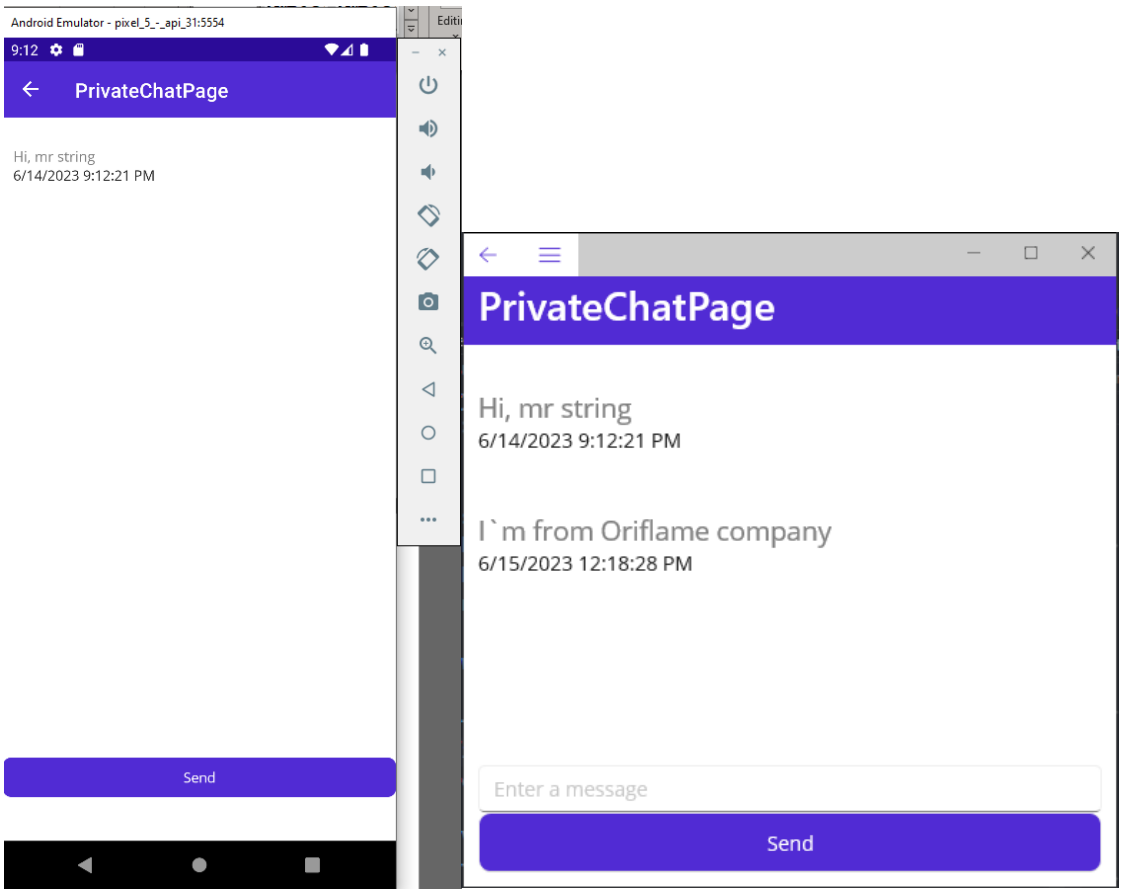

Рисунок 26 — Сторінка приватного чату на Android, рисунок 27 — Сторінка приватного чату на Windows

Для перевірки авторизованого користувача або виходу із аккаунта треба натиснути на позначку виїжджаючого вікна:

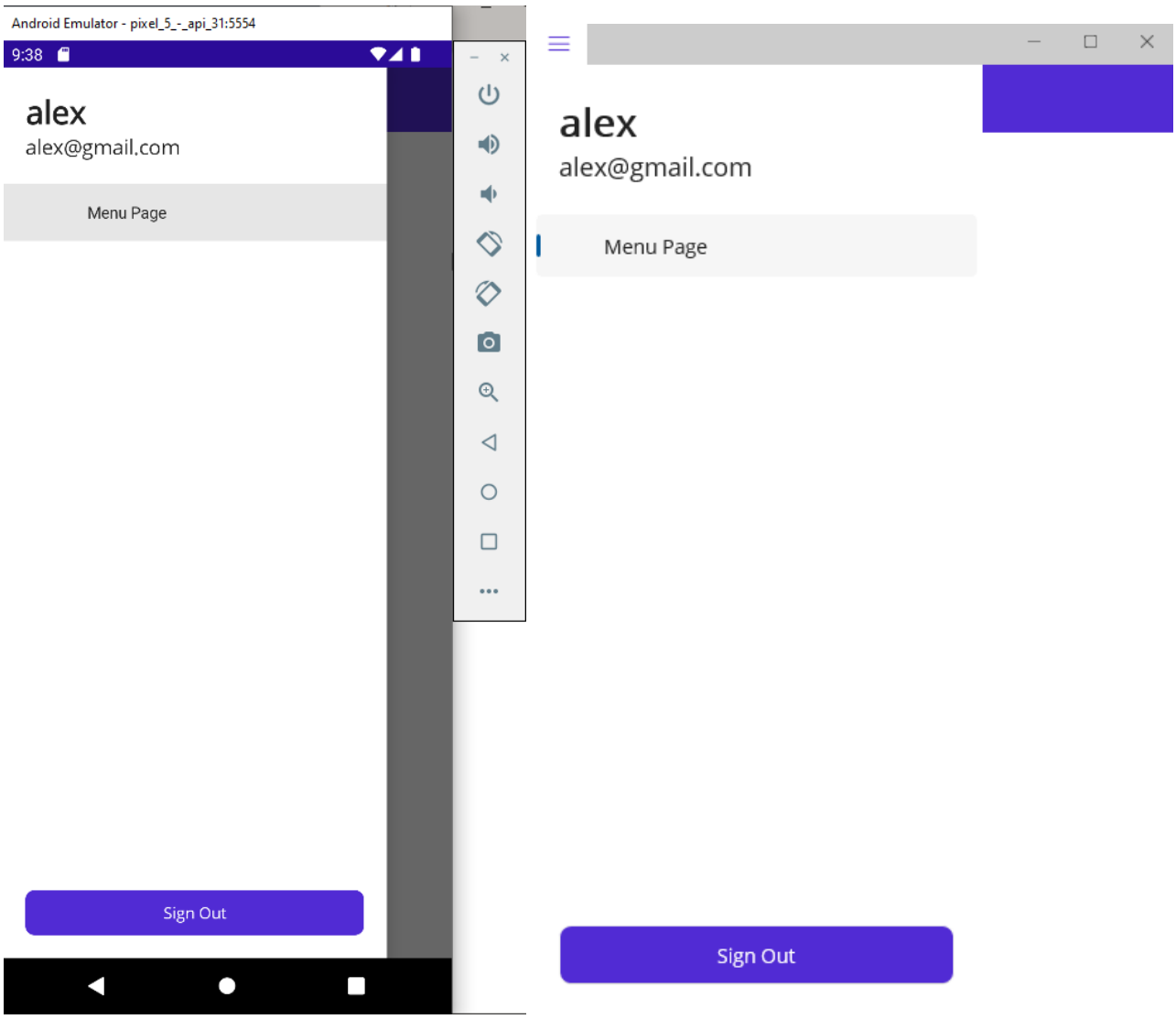

Рисунок 28 — Бічне вікно на Android, рисунок 29 — Бічне вікно на Windows

РОЗДІЛ 3

ЕКОНОМІЧНА ЧАСТИНА

Під час розробки програмного забезпечення важливими етапами є визначення трудомісткості розробки і розрахунок витрат на створення програмного продукту.

3.1. Визначення трудомісткості розробки програмного забезпечення

Задані дані:

1. передбачуване число операторів – 2850;
2. коефіцієнт складності програми – 1,9;
3. коефіцієнт корекції програми в ході її розробки – 0,3;
4. годинна заробітна плата програміста, грн/год – 150;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,1;
7. вартість машино-години ЕОМ, грн/год – 12.

Вартість машино-години ЕОМ було вирахувано наступним чином:

При роботі програми комп’ютер використовує 50 Ват на годину. Щоб визначити вартість однієї 50-ват-години за тарифом 2,8 грн кіловат-година, потрібно перевести тариф в ват-години.

1 кіловат-година = 1000Ват-година;

2,8 грн за кіловат-годину = 0.0028 грн за Ват-годину;

0.0028грн * 50Ват-годин = 0.14 гривень на енергію за годину роботи; (3.1)

Вартість комп’ютера складає 60к грн, при продажі ціна впаде вдвічі. В загалом комп’ютери смертельно застарівають за 5-10 років і її міняють. Виходячи з цього можна порахувати амортизацію лінійним методом:

(60000-30000)/7.5 = 4000 грн на рік = 333.33 грн на місяць = 0.0077 грн на секунду; (3.2)

Сумарна вартість моєї машино-години буде складати 0.0028 + 0.0077 = 0.0105 грн.

Але це без урахування придбання комп’ютера, приміщення, комунікацій ринкових умов. Тому фінальну ціну було підвищено до зразків, тобто до 12 грн/год.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин,} \quad (3.3)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q * C * (1 + p), \quad (3.4)$$

де q - передбачуване число операторів;

c - коефіцієнт складності програми;

p - коефіцієнт корекції програми в ході її розробки.

$$Q = 2850 * 1,9 * (1 + 0,3) = 7040$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q * B}{(75..85) * k}, \text{ людино-годин.} \quad (3.5)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

$$t_u = \frac{7040 * 1,3}{85 * 1,1} = \frac{9,151}{93,5} = 98, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25) * k}, \text{ людино-годин,} \quad (3.6)$$

$$t_a = \frac{7040}{20 * 1,1} = 320 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25) * k}, \text{ людино-годин,} \quad (3.7)$$

$$t_n = \frac{7040}{25 * 1{,}1} = 256, \text{ людино-годин,}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5) * k}, \text{ людино-годин,} \quad (3.8)$$

$$t_{отл} = \frac{7040}{5 * 1.1} = 1280 \text{ людино-годин;}$$

- за умови комплексного налагодження завдання:

$$t^k_{отл} = 1{,}5 * t_{отл}, \text{ людино-годин,} \quad (3.9)$$

$$t^k_{отл} = 1{,}5 * 1280 = 1920 \text{ людино-годин.}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}, \text{ людино-годин,} \quad (3.10)$$

де $t_{др}$ - трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{15..20 * k}, \text{ людино-годин,} \quad (3.11)$$

$$t_{др} = \frac{7040}{20 * 1.1} = 320 \text{ людино-годин.}$$

$t_{до}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{до} = 0{,}75 * t_{др}, \text{ людино-годин,} \quad (3.12)$$

$$t_{до} = 0{,}75 * 320 = 240 \text{ людино-годин},$$

$$t_{д} = 320 + 240 = 560 \text{ людино-годин}.$$

Тепер розрахуємо трудомісткість ПЗ:

$$t = 50 + 98 + 320 + 256 + 1280 + 560 = 2564 \text{ людино-годин}.$$

3.2. Витрати на створення програмного забезпечення

Витрати на створення ПЗ *Кпо* включають витрати на заробітну плату виконавця програми *Зз/п* і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$K_{по} = З_{зп} + З_{мв}, \text{ грн.} \quad (3.13)$$

Заробітна плата виконавців визначається за формулою:

$$З_{зп} = t * C_{пр}, \text{ грн}, \quad (3.14)$$

де t - загальна трудомісткість, людино-годин;

$C_{пр}$ - середня годинна заробітна плата програміста, грн/година.

$$З_{зп} = 2564 * 150 = 384600 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми:

$$З_{мв} = t_{отл} * C_{мч}, \text{ грн}, \quad (3.15)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год,

$C_{мч}$ - вартість машино-години ЕОМ, грн/год,

$C_{мч}$ = 15, грн/год.

$$З_{мв} = 1280 * 12 = 15360 \text{ грн.}$$

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення ПЗ:

$$К_{по} = 384600 + 15360 = 399960 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k * F_p}, \text{ міс.,} \quad (3.16)$$

де B_k - число виконавців (приймається 1),

F_p - місячний фонд робочого часу (40 годин на тиждень F_p = 176 годин).

$$Т = \frac{2564}{1 * 176} = 14{,}56 \text{ міс.}$$

Висновок:

Вартість даного продукту становить 399960 грн. і не вимагає додаткових витрат. Очікуваний час розробки становить 14,56 місяці. Цей термін пов'язаний зі значним числом операторів, і включає час на дослідження і розробку алгоритму вирішення поставленого завдання, програмування по готовому алгоритму, налагодження програми і підготовку документації.

ВИСНОВКИ

В рамках дипломного проекту був розроблений кросплатформний додаток для організації чату.

Це програмне забезпечення призначено для того, щоб користувач міг легко створювати та користуватися чатами, використовуючи при цьому зручний інтерфейс. Використання надає можливість користувачу швидко та ефективно обмінюватися інформацією з іншими людьми.

Під час виконання даного дипломного проекту були виконані наступні задачі:

- ✓ Проаналізовано предметну область поставленої задачі.
- ✓ Розроблено дизайн додатку.
- ✓ Спроектована внутрішня архітектура системи.
- ✓ Створено клієнтську частину системи.
- ✓ Створено локальне сховище даних.
- ✓ Визначено трудомісткість розробленої системи.
- ✓ Підрахована вартість додатку.

Розроблений веб-застосунок дозволяє:

- Використовувати клієнт на Android та Windows.
- Виконувати всі необхідні дії з чатом.
- Можливість зручної навігації між створеними середовищами додатку.
- Зберігати внесені дані.

Програмний продукт реалізований за допомогою мови програмування C# та XAML на фреймворках .NET та MAUI с СКБД MS SQL Server. MAUI було використано для реалізації клієнтської частини, .NET — для API, SQL Server — для збереження даних. Для візуальної складової клієнту був використаний XAML. Застосування кросплатформного рішення як MAUI може зекономити значну кількість часу та грошей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://en.wikipedia.org/wiki/ARPANET
2. https://learn.microsoft.com/uk-ua/dotnet/csharp/tour-of-csharp
3. https://aws-amazon-com.translate.goog/what-is/net/?_x_tr_sl=en&_x_tr_tl=ru&_x_tr_hl=ru&_x_tr_pto=rq#:~:text=NET%3F-,,and%20high-performance%20software%20development
4. http://edu.asu.in.ua/mod/book/view.php?id=117&chapterid=257
5. https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui
6. https://learn.microsoft.com/en-us/dotnet/maui/xaml/
7. https://uk.wikipedia.org/wiki/Microsoft_SQL_Server
8. https://learn.microsoft.com/en-us/ef/core/
9. https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures
10. https://learn.microsoft.com/ru-ru/dotnet/architecture/maui/mvvm

ДОДАТОК А

КОД ПРОГРАМИ

Код серверної частини

Program.cs

```
using KoalitionServer.Data;
using KoalitionServer.Models;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi.Models;
using Swashbuckle.AspNetCore.Filters;
using System.Text;
using MediatR;
using KoalitionServer.Services.UserServices;
using KoalitionServer.Services.GroupChatServices;
using KoalitionServer.Services.GroupMessagesServices;
using KoalitionServer.Services.PrivateChatServices;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddControllers();
builder.Services.AddMediatR(typeof(Program));
builder.Services.AddHttpContextAccessor();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddScoped<UserService>();
builder.Services.AddScoped<CreateGroupChatService>();
builder.Services.AddScoped<GetCurrentUserGroupChatsService>();
builder.Services.AddScoped<AddUserToGroupChatService>();
builder.Services.AddScoped<DeleteGroupChatorUserService>();
builder.Services.AddScoped<GroupMessageService>();
builder.Services.AddScoped<PrivateChatService>();
builder.Services.AddDbContext<AppDbContext>(options =>
{
   options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"))
   .EnableSensitiveDataLogging();
});


builder.Services.AddScoped<IPasswordHasher<User>, PasswordHasher<User>>();
builder.Services.AddSwaggerGen(options =>
{
   options.AddSecurityDefinition("oauth2", new OpenApiSecurityScheme
   {
      Description = "Standard Authorization header using the Bearer scheme (\"bearer {token}\")",
      In = ParameterLocation.Header,
      Name = "Authorization",
      Type = SecuritySchemeType.ApiKey
   });

   options.OperationFilter<SecurityRequirementsOperationFilter>();
});
```

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
   .AddJwtBearer(options =>
   {
      options.TokenValidationParameters = new TokenValidationParameters
      {
         ValidateIssuerSigningKey = true,
         IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8
            .GetBytes(builder.Configuration.GetSection("Jwt:Key").Value)),
         ValidateIssuer = false,
         ValidateAudience = false
      };
   });

builder.Services.AddCors(options => options.AddPolicy(name: "NgOrigins",
   policy =>
   {
      policy.WithOrigins("http://localhost:7127").AllowAnyMethod().AllowAnyHeader();
   }));

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
   app.UseSwagger();
   app.UseSwaggerUI();
}

app.UseCors("NgOrigins");
//app.UseHttpsRedirection();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllers();
app.UseSwagger(c =>
{
   c.RouteTemplate = "swagger/{documentName}/swagger.json";
});
app.UseSwaggerUI(options =>
{
   options.SwaggerEndpoint("/swagger/v1/swagger.json", "Koalition API");
   options.RoutePrefix = string.Empty;
});
app.Run();
```

appsettings.json

```
{
  "Jwt": {
    "Key": "my top secret key"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=KoalitionDb;Trusted_Connection=True;"
```

```
  },
  "AllowedHosts": "*"
}
```

UserService.cs

```
using KoalitionServer.Data;
using KoalitionServer.Models;
using KoalitionServer.Requests.UserRequests;
using KoalitionServer.Responses.UserResponses;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;

namespace KoalitionServer.Services.UserServices
{
    public class UserService
    {
        private readonly AppDbContext _context;
        private readonly IPasswordHasher<User> _passwordHasher;
        private readonly IConfiguration _configuration;
        private readonly IHttpContextAccessor _httpContextAccessor;

        public UserService(AppDbContext context, IPasswordHasher<User> passwordHasher, IConfiguration configuration,
IHttpContextAccessor httpContextAccessor)
        {
            _context = context;
            _passwordHasher = passwordHasher;
            _configuration = configuration;
            _httpContextAccessor = httpContextAccessor;
        }

        public async Task<User> RegisterUser(RegistrationRequest regRequest)
        {
            if (await _context.Users.AnyAsync(u => u.Email == regRequest.Email))
            {
                throw new ArgumentException("User with this email already exist!");
            }

            var newUser = new User
            {
                Login = regRequest.Login,
                Name = regRequest.Name,
                Email = regRequest.Email,
                Password = _passwordHasher.HashPassword(null, regRequest.Password)
            };

            _context.Users.Add(newUser);
            await _context.SaveChangesAsync();

            return newUser;
        }

        public async Task<User> UpdateUser(RegistrationRequest updateRequest)
        {
            var currentUser = _httpContextAccessor.HttpContext.User?.Claims?.FirstOrDefault(x => x.Type ==
ClaimTypes.Name)?.Value;
            var user = await _context.Users.FirstOrDefaultAsync(x => x.Login == currentUser);
            if (user == null)
```

```
    {
        throw new ArgumentException("Invalid login!");
    }
    user.Login = updateRequest.Login;
    user.Name = updateRequest.Name;
    user.Email = updateRequest.Email;
    user.Password = _passwordHasher.HashPassword(null, updateRequest.Password);
    _context.Users.Update(user);
    await _context.SaveChangesAsync();
    return user;
}


public async Task<AuthenticateResponse> Authenticate(AuthenticateRequest authRequest)
{
    var user = await _context.Users.FirstOrDefaultAsync(x => x.Login == authRequest.Login);
    if (user == null)
    {
        throw new ArgumentException("Invalid login!");
    }


    var result = _passwordHasher.VerifyHashedPassword(user, user.Password, authRequest.Password);
    if (result == PasswordVerificationResult.Failed)
    {
        throw new ArgumentException("Invalid password!");
    }

    var userDetails = new AuthenticatedUserResponse
    {
        UserId = user.UserId,
        Login = user.Login,
        Name = user.Name,
        Email = user.Email
    };
    string token = CreateToken(user);

    return new AuthenticateResponse { Token = token, UserDetails = userDetails };
}

private string CreateToken(User user)
{
    List<Claim> claims = new List<Claim>
    {
        new Claim(ClaimTypes.NameIdentifier, user.UserId.ToString()),
        new Claim(ClaimTypes.Name, user.Login),
        new Claim(ClaimTypes.Role, "Admin")
    };

    var key = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes(
        _configuration.GetSection("Jwt:Key").Value));

    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512Signature);

    var token = new JwtSecurityToken(
        claims: claims,
        expires: DateTime.Now.AddDays(1),
        signingCredentials: creds);

    var jwt = new JwtSecurityTokenHandler().WriteToken(token);

    return jwt;
```

```
    }

  }
}
```

DeleteCurrentUserService.cs

```
using KoalitionServer.Data;
using KoalitionServer.Requests.UserRequests;
using MediatR;
using Microsoft.EntityFrameworkCore;

namespace KoalitionServer.Services.UserServices
{
  public class DeleteCurrentUserService : IRequestHandler<DeleteCurrentUserRequest, bool>
  {
    private readonly AppDbContext _context;

    public DeleteCurrentUserService(AppDbContext context)
    {
      _context = context;
    }

    public async Task<bool> Handle(DeleteCurrentUserRequest request, CancellationToken cancellationToken)
    {
      var user = await _context.Users.FirstOrDefaultAsync(x => x.Login == request.Login, cancellationToken);

      _context.Users.Remove(user);
      await _context.SaveChangesAsync();

      return true;
    }
  }
}
```

CurrentUserService.cs

```
using KoalitionServer.Data;
using KoalitionServer.Requests.UserRequests;
using KoalitionServer.Responses.UserResponses;
using MediatR;
using Microsoft.EntityFrameworkCore;

namespace KoalitionServer.Services.UserServices
{
  public class CurrentUserService : IRequestHandler<CurrentUserRequestDTO, CurrentUserResponseDTO>
  {
    private readonly AppDbContext _context;

    public CurrentUserService(AppDbContext context)
    {
      _context = context;
    }

    public async Task<CurrentUserResponseDTO> Handle(CurrentUserRequestDTO request, CancellationToken
cancellationToken)
    {
```

```
        var user = await _context.Users.FirstOrDefaultAsync(x => x.Login == request.Login, cancellationToken);

        if (user == null)
        {
            throw new ArgumentException("Invalid login!");
        }

        return new CurrentUserResponseDTO
        {
            UserId = user.UserId,
            Login = user.Login,
            Name = user.Name,
            Email = user.Email
        };
    }
  }
}
```

PrivateChatService.cs

```
using KoalitionServer.Data;
using KoalitionServer.Models;
using KoalitionServer.Responses.GroupMessageResponses;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;

namespace KoalitionServer.Services.PrivateChatServices
{
  public class PrivateChatService
  {
    private readonly AppDbContext _context;
    private readonly IHttpContextAccessor _httpContextAccessor;

    public PrivateChatService(AppDbContext context, IHttpContextAccessor httpContextAccessor)
    {
      _context = context;
      _httpContextAccessor = httpContextAccessor;
    }

    public async Task<PrivateMessage> SendMessage(int recipientId, string messageText, ClaimsPrincipal user)
    {
      var senderId = Convert.ToInt32(user.FindFirst(x => x.Type == ClaimTypes.NameIdentifier)?.Value);
      var sender = await _context.Users.FindAsync(senderId);
      if (sender == null)
      {
        throw new ArgumentException("Sender not found.");
      }

      var recipient = await _context.Users.SingleOrDefaultAsync(u => u.UserId == recipientId);
      if (recipient == null)
      {
        throw new ArgumentException("Recipient not found.");
      }

      var privateChat = await _context.PrivateChats.SingleOrDefaultAsync(pc =>
        pc.PrivateChatsToUsers.Any(pcu => pcu.UserId == sender.UserId) &&
        pc.PrivateChatsToUsers.Any(pcu => pcu.UserId == recipient.UserId));

      if (privateChat == null)
      {
```

```
            privateChat = new PrivateChat();
            _context.PrivateChats.Add(privateChat);

            var privateChatsToUsers = new List<PrivateChatsToUsers>
            {
                new PrivateChatsToUsers {PrivateChat = privateChat, User = sender},
                new PrivateChatsToUsers {PrivateChat = privateChat, User = recipient},
            };
            _context.PrivateChatsToUsers.AddRange(privateChatsToUsers);
        }

        var privateMessage = new PrivateMessage
        {
            Text = messageText,
            Time = DateTime.Now,
            PrivateChat = privateChat,
            PrivateChatId = privateChat.PrivateChatId,
            UserId = sender.UserId
        };
        _context.PrivateMessages.Add(privateMessage);
        await _context.SaveChangesAsync();

        return privateMessage;
    }

    public async Task<List<ChatMessageResponse>> GetMessagesForPrivateChat(int recipientId)
    {
        var senderId = Convert.ToInt32(_httpContextAccessor.HttpContext.User?.Claims?.FirstOrDefault(x => x.Type ==
ClaimTypes.NameIdentifier)?.Value);
        var sender = await _context.Users.FindAsync(senderId);
        if (sender == null)
        {
            throw new ArgumentException("Sender not found.");
        }
        var recipient = await _context.Users.SingleOrDefaultAsync(u => u.UserId == recipientId);
        if (recipient == null)
        {
            throw new ArgumentException("Recipient not found.");
        }
        var privateChat = await _context.PrivateChats.SingleOrDefaultAsync(pc =>
            pc.PrivateChatsToUsers.Any(pcu => pcu.UserId == sender.UserId) &&
            pc.PrivateChatsToUsers.Any(pcu => pcu.UserId == recipient.UserId));
        if (privateChat == null)
        {
            throw new ArgumentException("Private chat not found.");
        }
        //add here
        var messages = await _context.PrivateMessages
            .Where(pm => pm.PrivateChatId == privateChat.PrivateChatId)
            .Select(pm => new ChatMessageResponse
            {
                Text = pm.Text,
                Time = pm.Time,
                UserId = pm.UserId
            })
            .ToListAsync();
        return messages;
    }

    public async Task UpdateMessage(int privateChatId, int messageId, string message)
    {
        var privateChat = await _context.PrivateChats
```

```
            .Include(gc => gc.PrivateChatsToUsers)
            .ThenInclude(gcu => gcu.User)
            .Include(gc => gc.Messages)
            .FirstOrDefaultAsync(gc => gc.PrivateChatId == privateChatId);
        if (privateChat == null)
            throw new ArgumentException($"Group chat with id {privateChatId} not found");

        var currentUser = _httpContextAccessor.HttpContext.User?.Claims?.FirstOrDefault(x => x.Type ==
ClaimTypes.NameIdentifier)?.Value;
        var privateChatUser = privateChat.PrivateChatsToUsers.FirstOrDefault(u => u.User.UserId ==
Convert.ToInt32(currentUser));
        if (privateChatUser == null)
            throw new ArgumentException("User is not a member of this group chat");

        var messageToUpdate = privateChat.Messages.FirstOrDefault(m => m.PrivateMessageId == messageId);
        if (messageToUpdate == null)
            throw new ArgumentException($"Message with id {messageId} not found");
        messageToUpdate.Text = message;
        await _context.SaveChangesAsync();
    }

    public async Task DeleteMessage(int privateChatId, int messageId)
    {
        var privateChat = await _context.PrivateChats
            .Include(gc => gc.PrivateChatsToUsers)
            .ThenInclude(gcu => gcu.User)
            .Include(gc => gc.Messages)
            .FirstOrDefaultAsync(gc => gc.PrivateChatId == privateChatId);
        if (privateChat == null)
            throw new ArgumentException($"Group chat with id {privateChatId} not found");

        var currentUser = _httpContextAccessor.HttpContext.User?.Claims?.FirstOrDefault(x => x.Type ==
ClaimTypes.NameIdentifier)?.Value;
        var privateChatUser = privateChat.PrivateChatsToUsers.FirstOrDefault(u => u.User.UserId ==
Convert.ToInt32(currentUser));
        if (privateChatUser == null)
            throw new ArgumentException("User is not a member of this group chat");

        var message = privateChat.Messages.FirstOrDefault(m => m.PrivateMessageId == messageId);
        if (message == null)
            throw new ArgumentException($"Message with id {messageId} not found");
        _context.PrivateMessages.Remove(message);
        await _context.SaveChangesAsync();
    }
  }
}
```

GroupMessageService.cs

```
using KoalitionServer.Data;
using KoalitionServer.Models;
using KoalitionServer.Responses.GroupMessageResponses;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;

namespace KoalitionServer.Services.GroupMessagesServices
{
  public class GroupMessageService
  {
    private readonly AppDbContext _context;
```

```
    private readonly IHttpContextAccessor _httpContextAccessor;

    public GroupMessageService(AppDbContext appDbContext, IHttpContextAccessor httpContextAccessor)
    {
        _context = appDbContext;
        _httpContextAccessor = httpContextAccessor;
    }

    public async Task SendGroupMessageAsync(int groupChatId, string message, ClaimsPrincipal user)
    {
        var groupChat = await _context.GroupChats
            .Include(gc => gc.GroupChatsToUsers)
            .ThenInclude(gcu => gcu.User)
            .FirstOrDefaultAsync(gc => gc.GroupChatId == groupChatId);

        if (groupChat == null)
        {
            throw new Exception($"Group chat with id '{groupChatId}' not found");
        }

        var sender = groupChat.GroupChatsToUsers
            .FirstOrDefault(u => u.User.UserId == Convert.ToInt32(user.FindFirst(x => x.Type ==
ClaimTypes.NameIdentifier)?.Value));

        var groupMessage = new GroupMessage
        {
            Text = message,
            Time = DateTime.Now,
            GroupChatId = groupChatId,
            UserId = sender.UserId
        };

        _context.GroupMessages.Add(groupMessage);
        await _context.SaveChangesAsync();
    }

    public async Task<List<ChatMessageResponse>> GetMessagesForGroupChat(int groupChatId)
    {
        var groupChat = await _context.GroupChats
            .Include(gc => gc.GroupChatsToUsers)
            .ThenInclude(gcu => gcu.User)
            .Include(gc => gc.Messages)
            .FirstOrDefaultAsync(gc => gc.GroupChatId == groupChatId);

        if (groupChat == null)
            throw new ArgumentException($"Group chat with id {groupChatId} not found");

        var currentUser = _httpContextAccessor.HttpContext.User?.Claims?.FirstOrDefault(x => x.Type ==
ClaimTypes.NameIdentifier)?.Value;
        var groupChatUser = groupChat.GroupChatsToUsers.FirstOrDefault(u => u.User.UserId ==
Convert.ToInt32(currentUser));
        if (groupChatUser == null)
            throw new ArgumentException("User is not a member of this group chat");

        var messages = groupChat.Messages
            .OrderBy(m => m.Time)
            .Select(m => new ChatMessageResponse
            {
                Text = m.Text,
                Time = m.Time,
                UserId = m.UserId
```

```
            })
            .ToList();

        return messages;
    }

    public async Task UpdateMessage(int groupChatId, int messageId, string message)
    {
        var groupChat = await _context.GroupChats
            .Include(gc => gc.GroupChatsToUsers)
            .ThenInclude(gcu => gcu.User)
            .Include(gc => gc.Messages)
            .FirstOrDefaultAsync(gc => gc.GroupChatId == groupChatId);
        if (groupChat == null)
            throw new ArgumentException($"Group chat with id {groupChatId} not found");
        var currentUser = _httpContextAccessor.HttpContext.User?.Claims?.FirstOrDefault(x => x.Type ==
ClaimTypes.NameIdentifier)?.Value;
        var groupChatUser = groupChat.GroupChatsToUsers.FirstOrDefault(u => u.User.UserId ==
Convert.ToInt32(currentUser));
        if (groupChatUser == null)
            throw new ArgumentException("User is not a member of this group chat");
        var messageToUpdate = groupChat.Messages.FirstOrDefault(m => m.GroupMessageId == messageId);
        if (messageToUpdate == null)
            throw new ArgumentException($"Message with id {messageId} not found");
        messageToUpdate.Text = message;
        await _context.SaveChangesAsync();
    }

    public async Task DeleteMessage(int groupChatId, int messageId)
    {
        var groupChat = await _context.GroupChats
            .Include(gc => gc.GroupChatsToUsers)
            .ThenInclude(gcu => gcu.User)
            .Include(gc => gc.Messages)
            .FirstOrDefaultAsync(gc => gc.GroupChatId == groupChatId);
        if (groupChat == null)
            throw new ArgumentException($"Group chat with id {groupChatId} not found");
        var currentUser = _httpContextAccessor.HttpContext.User?.Claims?.FirstOrDefault(x => x.Type ==
ClaimTypes.NameIdentifier)?.Value;
        var groupChatUser = groupChat.GroupChatsToUsers.FirstOrDefault(u => u.User.UserId ==
Convert.ToInt32(currentUser));
        if (groupChatUser == null)
            throw new ArgumentException("User is not a member of this group chat");
        var message = groupChat.Messages.FirstOrDefault(m => m.GroupMessageId == messageId);
        if (message == null)
            throw new ArgumentException($"Message with id {messageId} not found");
        _context.GroupMessages.Remove(message);
        await _context.SaveChangesAsync();
    }

  }
}
```

GetCurrentUserGroupChatsService.cs

```
using KoalitionServer.Data;
using KoalitionServer.Responses.GroupChatResponses;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;
```

```
namespace KoalitionServer.Services.GroupChatServices
{
  public class GetCurrentUserGroupChatsService
  {
    private readonly AppDbContext _context;
    private readonly IHttpContextAccessor _httpContextAccessor;

    public GetCurrentUserGroupChatsService(AppDbContext context, IHttpContextAccessor httpContextAccessor)
    {
      _context = context;
      _httpContextAccessor = httpContextAccessor;
    }

    public async Task<List<GroupChatResponse>> GetCurrentUserGroupChats()
    {
      var currentUserLogin = _httpContextAccessor.HttpContext.User?.Claims?.FirstOrDefault(x => x.Type ==
ClaimTypes.Name)?.Value;

      var groupChats = await _context.GroupChats
        .Where(gc => gc.GroupChatsToUsers.Any(gcu => gcu.User.Login == currentUserLogin))
        .ToListAsync();

      var groupChatDtos = groupChats.Select(gc => new GroupChatResponse
      {
        Id = gc.GroupChatId,
        Name = gc.Name
      }).ToList();

      return groupChatDtos;
    }
  }
}
```

DeleteGroupChatorUserService.cs

```
using KoalitionServer.Data;
using KoalitionServer.Requests.GroupChatRequests;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;

namespace KoalitionServer.Services.GroupChatServices
{
  public class DeleteGroupChatorUserService
  {
    private readonly AppDbContext _context;

    public DeleteGroupChatorUserService(AppDbContext context)
    {
      _context = context;
    }

    public async Task UpdateGroupChat(string groupName, CreateGroupChatRequest update, ClaimsPrincipal user)
    {
      var groupChat = await _context.GroupChats
        .Include(gc => gc.GroupChatsToUsers)
        .ThenInclude(gtu => gtu.User)
        .FirstOrDefaultAsync(gc => gc.Name == groupName);

      if (groupChat == null)
```

```
    {
        throw new Exception($"Group chat with name '{groupName}' not found");
    }

    var isOwner = groupChat.GroupChatsToUsers
        .FirstOrDefault(gtu => gtu.User.Login == user.FindFirst(ClaimTypes.Name).Value && gtu.IsOwner);

    if (isOwner == null)
    {
        throw new Exception("You are not authorized to delete this group chat");
    }

    groupChat.Name = update.Name;
    groupChat.Description = update.Description;
    await _context.SaveChangesAsync();
}

public async Task DeleteGroupChatAsync(string groupName, ClaimsPrincipal user)
{
    var groupChat = await _context.GroupChats
        .Include(gc => gc.GroupChatsToUsers)
        .ThenInclude(gtu => gtu.User)
        .FirstOrDefaultAsync(gc => gc.Name == groupName);

    if (groupChat == null)
    {
        throw new Exception($"Group chat with name '{groupName}' not found");
    }

    var isOwner = groupChat.GroupChatsToUsers
        .FirstOrDefault(gtu => gtu.User.Login == user.FindFirst(ClaimTypes.Name).Value && gtu.IsOwner);

    if (isOwner == null)
    {
        throw new Exception("You are not authorized to delete this group chat");
    }

    _context.GroupChats.Remove(groupChat);
    await _context.SaveChangesAsync();
}

public async Task<bool> DeleteGroupChatMemberAsync(int groupChatId, int userId, ClaimsPrincipal user)
{
    var groupChat = await _context.GroupChats
        .Include(gc => gc.GroupChatsToUsers)
        .ThenInclude(gcu => gcu.User)
        .FirstOrDefaultAsync(gc => gc.GroupChatId == groupChatId);

    if (groupChat == null)
    {
        return false;
    }

    var isOwner = groupChat.GroupChatsToUsers
        .FirstOrDefault(gtu => gtu.User.Login == user.FindFirst(ClaimTypes.Name).Value && gtu.IsOwner);

    if (isOwner == null)
    {
        throw new Exception("You are not authorized to delete this group chat");
    }

    var userToDelete = await _context.Users.FirstOrDefaultAsync(u => u.UserId == userId);
```

```
        var groupChatToUserToDelete = groupChat.GroupChatsToUsers.FirstOrDefault(gcu => gcu.UserId ==
userToDelete.UserId);

        _context.GroupChatsToUsers.Remove(groupChatToUserToDelete);
        await _context.SaveChangesAsync();
        return true;
      }

   }
}
```

CreateGroupChatService.cs

```
using KoalitionServer.Data;
using KoalitionServer.Models;
using KoalitionServer.Requests.GroupChatRequests;
using MediatR;
using Microsoft.EntityFrameworkCore;
using System.Security.Claims;

namespace KoalitionServer.Services.GroupChatServices
{
   public class CreateGroupChatService : IRequestHandler<CreateGroupChatRequest, string>
   {
      private readonly AppDbContext _context;
      private readonly IHttpContextAccessor _httpContextAccessor;

      public CreateGroupChatService(AppDbContext context, IHttpContextAccessor httpContextAccessor)
      {
         _context = context;
         _httpContextAccessor = httpContextAccessor;
      }

      public async Task<string> Handle(CreateGroupChatRequest request, CancellationToken cancellationToken)
      {
         var login = _httpContextAccessor.HttpContext.User?.Claims?.FirstOrDefault(x => x.Type == ClaimTypes.Name)?.Value;
         if (login == null)
         {
            throw new InvalidOperationException("User not found");
         }

         var user = await _context.Users.FirstOrDefaultAsync(l => l.Login == login, cancellationToken);
         if (user == null)
         {
            throw new InvalidOperationException("User not found");
         }

         var chat = new GroupChat
         {
            Name = request.Name,
            Description = request.Description
         };

         _context.GroupChats.Add(chat);

         var groupChatUser = new GroupChatsToUsers
         {
            GroupChat = chat,
            User = user,
            IsOwner = true
```

```
            };

            _context.GroupChatsToUsers.Add(groupChatUser);

            await _context.SaveChangesAsync();

            return chat.Name;
        }
    }
}
```

AddUserToGroupChatService.cs

```
using KoalitionServer.Data;
using KoalitionServer.Models;
using KoalitionServer.Requests.GroupChatRequests;
using MediatR;
using Microsoft.EntityFrameworkCore;

namespace KoalitionServer.Services.GroupChatServices
{
    public class AddUserToGroupChatService : IRequestHandler<AddUserToGroupChatRequest, bool>
    {
        private readonly AppDbContext _context;

        public AddUserToGroupChatService(AppDbContext context)
        {
            _context = context;
        }

        public async Task<bool> Handle(AddUserToGroupChatRequest request, CancellationToken cancellationToken)
        {
            var chatId = request.GroupChatId;
            var userId = request.UserId;

            var chat = await _context.GroupChats
                .Include(c => c.GroupChatsToUsers)
                .ThenInclude(gctu => gctu.User)
                .FirstOrDefaultAsync(c => c.GroupChatId == chatId, cancellationToken);

            if (chat == null)
            {
                throw new InvalidOperationException("Chat not found");
            }

            if (!chat.GroupChatsToUsers.Any(gctu => gctu.UserId == userId && gctu.GroupChatId == chatId))
            {
                chat.GroupChatsToUsers.Add(new GroupChatsToUsers
                {
                    GroupChatId = chatId,
                    UserId = userId,
                    IsOwner = false
                });

                await _context.SaveChangesAsync();
            }

            return true;
        }
    }
}
```

CurrentUserResponseDTO.cs

```
namespace KoalitionServer.Responses.UserResponses
{
  public class CurrentUserResponseDTO
  {
    public int UserId { get; set; }
    public string Login { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
  }
}
```

CurrentUserResponse.cs

```
namespace KoalitionServer.Responses.UserResponses
{
  public class CurrentUserResponse
  {
    public class UserContainer
    {
      public int UserId { get; set; }
      public string Login { get; set; }
      public string Name { get; set; }
      public string Email { get; set; }
    }

    public UserContainer User { get; set; }
  }
}
```

AuthenticateResponse.cs

```
namespace KoalitionServer.Responses.UserResponses
{
  public class AuthenticateResponse
  {
    public string Token { get; set; }
    public AuthenticatedUserResponse UserDetails { get; set; }
  }
}
```

AuthenticatedUserResponse.cs

```
namespace KoalitionServer.Responses.UserResponses
{
  public class AuthenticatedUserResponse
  {
    public int UserId { get; set; }
    public string Login { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
  }
}
```

PrivateMessageResponse.cs

```
namespace KoalitionServer.Responses.PrivateChatResponses
{
  public class PrivateMessageResponse
  {
    public string Text { get; set; }
    public int SenderId { get; set; }
    public DateTime Time { get; set; }
  }
}
```

SendMessageResponse.cs

```
namespace KoalitionServer.Responses.GroupMessageResponses
{
  public class SendMessageResponse
  {
    public string Text { get; set; }
    public DateTime Time { get; set; }
  }
}
```

ChatMessageResponse.cs

```
namespace KoalitionServer.Responses.GroupMessageResponses
{
  public class ChatMessageResponse
  {
    public string Text { get; set; }
    public DateTime Time { get; set; }
    public int UserId { get; set; }
  }
}
```

GroupChatResponse.cs

```
namespace KoalitionServer.Responses.GroupChatResponses
{
  public class GroupChatResponse
  {
    public int Id { get; set; }
    public string Name { get; set; }
  }
}
```

RegistrationRequest.cs

```
namespace KoalitionServer.Requests.UserRequests
{
  public class RegistrationRequest
  {
    public string Login { get; set; } = null!;
    public string Name { get; set; } = null!;
    public string Email { get; set; } = null!;
    public string Password { get; set; } = null!;
  }
```

```
}
```

DeleteCurrentUserRequest.cs

```
using MediatR;

namespace KoalitionServer.Requests.UserRequests
{
  public class DeleteCurrentUserRequest : IRequest<bool>
  {
    public string Login { get; set; }

    public DeleteCurrentUserRequest(string login)
    {
      Login = login;
    }
  }

}
```

CurrentUserRequestDTO.cs

```
using KoalitionServer.Responses.UserResponses;
using MediatR;

namespace KoalitionServer.Requests.UserRequests
{
  public class CurrentUserRequestDTO : IRequest<CurrentUserResponseDTO>
  {
    public CurrentUserRequestDTO(string login)
    {
      Login = login;
    }
    public string Login { get; set; }
  }
}
```

AuthenticateRequest.cs

```
namespace KoalitionServer.Requests.UserRequests
{
  public class AuthenticateRequest
  {
    public string Login { get; set; }
    public string Password { get; set; }
  }
}
```

PrivateMessageRequest.cs

```
namespace KoalitionServer.Requests.PrivateChatRequests
{
  public class PrivateMessageRequest
  {
    public string Text { get; set; }
  }
}
```

SendMessageRequest.cs

```
namespace KoalitionServer.Requests.GroupMessageRequests
{
  public class SendMessageRequest
  {
    public string Text { get; set; }
  }
}
```

CreateGroupChatRequest.cs

```
using MediatR;

namespace KoalitionServer.Requests.GroupChatRequests
{
  public class CreateGroupChatRequest:IRequest<string>
  {
    public string Name { get; set; }
    public string Description { get; set; }
  }
}
```

AddUserToGroupChatRequest.cs

```
using MediatR;

namespace KoalitionServer.Requests.GroupChatRequests
{
  public class AddUserToGroupChatRequest : IRequest<bool>
  {
    public int GroupChatId { get; set; }
    public int UserId { get; set; }
  }
}
```

User.cs

```
namespace KoalitionServer.Models
{
  public class User
  {
    public int UserId { get; set; }
    public string Login { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
    public bool? Online { get; set; }
    public DateTime? LastRescent { get; set; }

    //one user to many messages and groups
    public ICollection<PrivateMessage> PrivateMessages { get; set; }
    public ICollection<GroupMessage> GroupMessages { get; set; }
    public ICollection<GroupChatsToUsers> GroupChatsToUsers { get; set; }
    public ICollection<PrivateChat> PrivateChats { get; set; }
    public ICollection<PrivateChatsToUsers> PrivateChatsToUsers { get; set; }
```

```
  }
}
```

PrivateMessage.cs

```
namespace KoalitionServer.Models
{
  public class PrivateMessage
  {
    public int PrivateMessageId { get; set; }
    public string Text { get; set; }
    public DateTime Time { get; set; }
    public int PrivateChatId { get; set; }
    public PrivateChat PrivateChat { get; set; }
    public int UserId { get; set; }
    public PrivateMessage()
    {
      Time = DateTime.Now;
    }
  }
}
```

PrivateChatsToUsers.cs

```
namespace KoalitionServer.Models
{
  public class PrivateChatsToUsers
  {
    public int PrivateChatId { get; set; }
    public PrivateChat PrivateChat { get; set; }
    public int UserId { get; set; }
    public User User { get; set; }
  }
}
```

PrivateChat.cs

```
namespace KoalitionServer.Models
{
  public class PrivateChat
  {
    public int PrivateChatId { get; set; }
    public ICollection<PrivateMessage> Messages { get; set; }
    public ICollection<PrivateChatsToUsers> PrivateChatsToUsers { get; set; }
  }
}
```

GroupMessage.cs

```
namespace KoalitionServer.Models
{
  public class GroupMessage
  {
    public int GroupMessageId { get; set; }
    public string Text { get; set; }
```

```
        public DateTime Time { get; set; }
        public int GroupChatId { get; set; }
        public int UserId { get; set; }
        public GroupMessage()
        {
            Time = DateTime.Now;
        }
    }
}
```

GroupChatsToUsers.cs

```
namespace KoalitionServer.Models
{
    public class GroupChatsToUsers
    {
        public int GroupChatId { get; set; }
        public GroupChat GroupChat { get; set; }
        public int UserId { get; set; }
        public User User { get; set; }
        public bool IsOwner { get; set; }
    }
}
```

GroupChat.cs

```
namespace KoalitionServer.Models
{
    public class GroupChat
    {
        public int GroupChatId { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public ICollection<GroupMessage> Messages { get; set; }
        public ICollection<GroupChatsToUsers> GroupChatsToUsers { get; set; }
    }
}
```

ErrorViewModel.cs

```
namespace KoalitionServer.Models
{
    public class ErrorViewModel
    {
        public string? RequestId { get; set; }

        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}
```

AppDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using KoalitionServer.Models;

namespace KoalitionServer.Data
```

```
{
  public class AppDbContext : DbContext
  {
    public AppDbContext(DbContextOptions<AppDbContext> options): base(options)
      { }

    public DbSet<GroupMessage> GroupMessages { get; set; }
    public DbSet<PrivateMessage> PrivateMessages { get; set; }
    public DbSet<User> Users { get; set; }
    public DbSet<GroupChat> GroupChats{ get; set; }
    public DbSet<PrivateChat> PrivateChats { get; set; }
    public DbSet<GroupChatsToUsers> GroupChatsToUsers { get; set;}
    public DbSet<PrivateChatsToUsers> PrivateChatsToUsers { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
      modelBuilder.Entity<GroupChatsToUsers>()
        .HasKey(gcu => new { gcu.GroupChatId, gcu.UserId });

      modelBuilder.Entity<GroupChatsToUsers>()
        .HasOne(gcu => gcu.GroupChat)
        .WithMany(g => g.GroupChatsToUsers)
        .HasForeignKey(gcu => gcu.GroupChatId);

      modelBuilder.Entity<GroupChatsToUsers>()
        .HasOne(gcu => gcu.User)
        .WithMany(u => u.GroupChatsToUsers)
        .HasForeignKey(gcu => gcu.UserId);

      modelBuilder.Entity<GroupChatsToUsers>()
        .Property(gcu => gcu.IsOwner)
        .IsRequired();



      modelBuilder.Entity<PrivateChatsToUsers>()
        .HasKey(pcu => new { pcu.PrivateChatId, pcu.UserId });

      modelBuilder.Entity<PrivateChatsToUsers>()
        .HasOne(pcu => pcu.PrivateChat)
        .WithMany(p => p.PrivateChatsToUsers)
        .HasForeignKey(pcu => pcu.PrivateChatId);

      modelBuilder.Entity<PrivateChatsToUsers>()
        .HasOne(pcu => pcu.User)
        .WithMany(u => u.PrivateChatsToUsers)
        .HasForeignKey(pcu => pcu.UserId);
    }
  }
}
```

UsersController.cs

```
using Microsoft.AspNetCore.Mvc;
using KoalitionServer.Requests.UserRequests;
using KoalitionServer.Models;
using Microsoft.AspNetCore.Authorization;
using KoalitionServer.Responses.UserResponses;
using System.Security.Claims;
using Microsoft.EntityFrameworkCore;
using KoalitionServer.Data;
```

```
using MediatR;
using KoalitionServer.Services.UserServices;

namespace KoalitionServer.Controllers
{
   [ApiController]
   [Route("api/[controller]")]
   public class UsersController : ControllerBase
   {
      private readonly IMediator _mediator;
      private readonly UserService _userService;
      private readonly AppDbContext _context;
      private readonly IHttpContextAccessor _httpContextAccessor;

      public UsersController(UserService userService, AppDbContext appDbContext, IMediator mediator,
IHttpContextAccessor httpContextAccessor)
      {
         _userService = userService;
         _context = appDbContext;
         _mediator = mediator;
         _httpContextAccessor = httpContextAccessor;
      }

      [HttpGet("allUsers"), Authorize]
      public async Task<ActionResult<List<User>>> GetAllUsers()
      {
         var users = await _context.Users.ToListAsync();
         return Ok(users);
      }

      [HttpGet("currentuser"), Authorize]
      public async Task<CurrentUserResponse> GetCurrentAuthenticatedUser()
      {
         var user = await _mediator.Send(new CurrentUserRequestDTO
            (_httpContextAccessor.HttpContext.User?.Claims?.FirstOrDefault(x => x.Type == ClaimTypes.Name)?.Value));

         return new CurrentUserResponse()
         {
            User = new CurrentUserResponse.UserContainer()
            {
               UserId = user.UserId,
               Login = user.Login,
               Name = user.Name,
               Email = user.Email
            }
         };
      }

      [HttpPost("register")]
      public async Task<ActionResult<User>> RegisterUser([FromBody] RegistrationRequest regRequest)
      {
         var user = await _userService.RegisterUser(regRequest);
         return Ok(user);
      }

      [HttpPut("update"), Authorize]
      public async Task<ActionResult<User>> UpdateUser([FromBody] RegistrationRequest updateRequest)
      {
         var user = await _userService.UpdateUser(updateRequest);
         return Ok(user);
      }
```

```
    [HttpPost("login")]
    public async Task<ActionResult<AuthenticateResponse>> Authenticate([FromBody] AuthenticateRequest authRequest)
    {
      var currentUserResponse = await _userService.Authenticate(authRequest);
      return Ok(currentUserResponse);
    }

    [HttpDelete("currentuser"), Authorize]
    public async Task<bool> DeleteCurrentUser()
    {
      return await _mediator.Send(new DeleteCurrentUserRequest
        (_httpContextAccessor.HttpContext.User?.Claims?.FirstOrDefault(x => x.Type == ClaimTypes.Name)?.Value));
    }


  }
}
```

PrivateChatsController.cs

```
using KoalitionServer.Requests.PrivateChatRequests;
using KoalitionServer.Responses.GroupMessageResponses;
using KoalitionServer.Services.PrivateChatServices;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace KoalitionServer.Controllers
{
  [Route("api/[controller]")]
  [ApiController]
  [Authorize]
  public class PrivateChatsController : ControllerBase
  {
    private readonly PrivateChatService _privateChatService;

    public PrivateChatsController(PrivateChatService privateChatService)
    {
      _privateChatService = privateChatService;
    }

    [HttpGet]
    [Authorize]
    public async Task<ActionResult<IEnumerable<ChatMessageResponse>>> GetGroupChatMessages(int recipientId)
    {
      var messages = await _privateChatService.GetMessagesForPrivateChat(recipientId);

      var messageDtos = new List<ChatMessageResponse>();
      foreach (var message in messages)
      {
        messageDtos.Add(new ChatMessageResponse
        {
          Text = message.Text,
          Time = message.Time,
          UserId = message.UserId,
        });
      }

      return Ok(messageDtos);
    }

    [HttpPost("sendMessage")]
```

```
    [Authorize]
    public async Task<ActionResult<SendMessageResponse>> SendGroupMessage(int recipientId, [FromBody]
PrivateMessageRequest message)
    {
      await _privateChatService.SendMessage(recipientId, message.Text, User);

      var responseDto = new SendMessageResponse
      {
        Text = message.Text,
        Time = DateTime.Now,
      };

      return Ok(responseDto);
    }

    [HttpPut("{messageId}")]
    [Authorize]
    public async Task<IActionResult> UpdateGroupMessage(int privateChatId, int messageId, [FromBody]
PrivateMessageRequest message)
    {
      await _privateChatService.UpdateMessage(privateChatId, messageId, message.Text);
      return NoContent();
    }

    [HttpDelete("{messageId}")]
    [Authorize]
    public async Task<IActionResult> DeleteGroupMessage(int privateChatId, int messageId)
    {
      await _privateChatService.DeleteMessage(privateChatId, messageId);
      return NoContent();
    }
  }
}
```

GroupMessagesController.cs

```
using KoalitionServer.Services.GroupMessagesServices;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using KoalitionServer.Requests.GroupMessageRequests;
using KoalitionServer.Responses.GroupMessageResponses;

namespace KoalitionServer.Controllers
{
  [ApiController]
  [Route("api/groupchats/{groupChatId}/messages")]
  public class GroupMessagesController : ControllerBase
  {
    private readonly GroupMessageService _groupMessageService;

    public GroupMessagesController(GroupMessageService groupMessageService)
    {
      _groupMessageService = groupMessageService;
    }

    [HttpPost("sendMessage")]
    [Authorize]
    public async Task<ActionResult<SendMessageResponse>> SendGroupMessage(int groupChatId, [FromBody]
SendMessageRequest message)
    {
      await _groupMessageService.SendGroupMessageAsync(groupChatId, message.Text, User);
```

```
            var responseDto = new SendMessageResponse
            {
                Text = message.Text,
                Time = DateTime.Now,
            };

            return Ok(responseDto);
        }

        [HttpGet]
        [Authorize]
        public async Task<ActionResult<IEnumerable<ChatMessageResponse>>> GetGroupChatMessages(int groupChatId)
        {
            var messages = await _groupMessageService.GetMessagesForGroupChat(groupChatId);

            var messageDtos = new List<ChatMessageResponse>();
            foreach (var message in messages)
            {
                messageDtos.Add(new ChatMessageResponse
                {
                    Text = message.Text,
                    Time = message.Time,
                    UserId = message.UserId,
                });
            }

            return Ok(messageDtos);
        }

        [HttpPut("{messageId}")]
        [Authorize]
        public async Task<IActionResult> UpdateGroupMessage(int groupChatId, int messageId, [FromBody]
SendMessageRequest message)
        {
            await _groupMessageService.UpdateMessage(groupChatId, messageId, message.Text);
            return NoContent();
        }

        [HttpDelete("{messageId}")]
        [Authorize]
        public async Task<IActionResult> DeleteGroupMessage(int groupChatId, int messageId)
        {
            await _groupMessageService.DeleteMessage(groupChatId, messageId);
            return NoContent();
        }

    }



}
```

GroupChatController.cs

```
using KoalitionServer.Requests.GroupChatRequests;
using KoalitionServer.Services.GroupChatServices;
using MediatR;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
```

```
namespace KoalitionServer.Controllers
{
   [Route("api/[controller]")]
   [ApiController]
   public class GroupChatController : ControllerBase
   {
      private readonly IMediator _mediator;
      private readonly GetCurrentUserGroupChatsService _getCurrentUserGroupChatsService;
      private readonly DeleteGroupChatorUserService _deleteGroupChatorUserService;

      public GroupChatController(IMediator mediator, GetCurrentUserGroupChatsService getAllGroupChatsService,
DeleteGroupChatorUserService deleteGroupChatorUserService)
      {
         _mediator = mediator;
         _getCurrentUserGroupChatsService = getAllGroupChatsService;
         _deleteGroupChatorUserService = deleteGroupChatorUserService;
      }

      [HttpPost("createGroupChat")]
      [Authorize]
      public async Task<ActionResult<string>> CreateGroupChat(CreateGroupChatRequest request)
      {
         var chatName = await _mediator.Send(request);
         return chatName;
      }

      [HttpPost("addUser")]
      [Authorize]
      public async Task<ActionResult<bool>> AddUserToGroupChat(AddUserToGroupChatRequest request)
      {
         var result = await _mediator.Send(request);
         return result;
      }

      [HttpGet("getChats")]
      [Authorize]
      public async Task<IActionResult> GetCurrentUserGroupChats()
      {
         var groupChatDtos = await _getCurrentUserGroupChatsService.GetCurrentUserGroupChats();
         return Ok(groupChatDtos);
      }

      [HttpPut("updateGroupChat")]
      [Authorize]
      public async Task<IActionResult> UpdateGroupChat(string groupName, [FromBody] CreateGroupChatRequest update)
      {
         await _deleteGroupChatorUserService.UpdateGroupChat(groupName, update, User);
         return NoContent();
      }


      [HttpDelete("{groupName}")]
      [Authorize]
      public async Task<IActionResult> DeleteGroupChat(string groupName)
      {
         await _deleteGroupChatorUserService.DeleteGroupChatAsync(groupName, User);
         return NoContent();
      }

      [HttpDelete("deleteUserFromChat")]
      [Authorize]
      public async Task<IActionResult> DeleteGroupChatMember(int groupChatId, int userId)
```

```
    {
        var deleted = await _deleteGroupChatorUserService.DeleteGroupChatMemberAsync(groupChatId, userId, User);
        if (!deleted)
        {
            return NotFound();
        }

        return NoContent();
    }
  }
}
```

Код клієнтської частини

MauiProgram.cs

```
using KoalitionAndroidClient.Models;
using KoalitionAndroidClient.Services;
using KoalitionAndroidClient.ViewModels.AddGroupChatPageViewModel;
using KoalitionAndroidClient.ViewModels.Chat;
using KoalitionAndroidClient.ViewModels.Menu;
using KoalitionAndroidClient.ViewModels.Startup;
using KoalitionAndroidClient.Views.AddGroupChatPage;
using KoalitionAndroidClient.Views.Chat;
using KoalitionAndroidClient.Views.Menu;
using KoalitionAndroidClient.Views.Startup;

namespace KoalitionAndroidClient
{
  public static class MauiProgram
  {
    public static MauiApp CreateMauiApp()
    {
      var builder = MauiApp.CreateBuilder();
      builder
        .UseMauiApp<App>()
        .ConfigureFonts(fonts =>
        {
          fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
          fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
        });

      builder.Services.AddSingleton<ILoginService, LoginService>();
      builder.Services.AddSingleton<MenuPageViewModel>();

      builder.Services.AddSingleton<LoginPage>();
      builder.Services.AddSingleton<MenuPage>();
      builder.Services.AddSingleton<GroupChatPage>();
      builder.Services.AddSingleton<LoginResponse>();
      builder.Services.AddSingleton<ChatMessageResponse>();
      builder.Services.AddSingleton<UserBasicInfo>();
      builder.Services.AddSingleton<AddGroupChatPage>();
      builder.Services.AddSingleton<GroupChatPage>();

      builder.Services.AddSingleton<LoginPageViewModel>();
      builder.Services.AddSingleton<MenuPageViewModel>();
      builder.Services.AddSingleton<GroupChatResponse>();
      builder.Services.AddSingleton<GroupChatPageViewModel>();
      builder.Services.AddSingleton<GroupChatPageViewModel>();
      builder.Services.AddSingleton<AddGroupChatPageViewModel>();
```

```
        return builder.Build();
    }
  }
}
```

AppShell.xaml

```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
  x:Class="KoalitionAndroidClient.AppShell"
  xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:KoalitionAndroidClient"
  xmlns:models="clr-namespace:KoalitionAndroidClient.Models"
  xmlns:menuPages="clr-namespace:KoalitionAndroidClient.Views.Menu"
  xmlns:startupPages="clr-namespace:KoalitionAndroidClient.Views.Startup"
  xmlns:chat="clr-namespace:KoalitionAndroidClient.Views.Chat"
  xmlns:addChatPage="clr-namespace:KoalitionAndroidClient.Views.AddGroupChatPage">


  <ShellContent
    Title="Login Page" FlyoutItemIsVisible="False" Shell.FlyoutBehavior="Disabled"
    ContentTemplate="{DataTemplate startupPages:LoginPage}"
    Route="LoginPage" />

  <ShellContent
    Title="Menu Page"
    ContentTemplate="{DataTemplate menuPages:MenuPage}"
    Route="MenuPage" />

  <ShellContent
    Title="GroupChatPage" FlyoutItemIsVisible="False"
    ContentTemplate="{DataTemplate chat:GroupChatPage}"
    Route="GroupChatPage" />

  <ShellContent
    Title="AddGroupChatPage" FlyoutItemIsVisible="False"
    ContentTemplate="{DataTemplate addChatPage:AddGroupChatPage}"
    Route="AddGroupChatPage" />

  <ShellContent
    Title="PrivateChatPage" FlyoutItemIsVisible="False"
    ContentTemplate="{DataTemplate chat:PrivateChatPage}"
    Route="PrivateChatPage" />


  <Shell.FlyoutFooter>
    <StackLayout Padding="20">
      <Button Text="Sign Out" Command="{Binding SignOutCommand}" />
    </StackLayout>
  </Shell.FlyoutFooter>

</Shell>
```

LoginPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="KoalitionAndroidClient.Views.Startup.LoginPage"
```

```
        Title="LoginPage">

    <VerticalStackLayout
        Spacing="25"
        Padding="30,0"
        VerticalOptions="Center">

        <Label
            Text="Wellcome to Koalition!"
            SemanticProperties.HeadingLevel="Level1"
            FontSize="32"
            HorizontalOptions="Center" />

        <Frame HeightRequest="45" Margin="-20,0,0,0" Padding="0" HasShadow="True" BorderColor="White"
HorizontalOptions="FillAndExpand">
            <Entry Text="{Binding Login}" Margin="20,0,0,0" VerticalOptions="Center" Placeholder="User Name"
Keyboard="Email" />
        </Frame>

        <Frame HeightRequest="45" Margin="-20,0,0,0" Padding="0" HasShadow="True" BorderColor="White"
HorizontalOptions="FillAndExpand">
            <Entry Text="{Binding Password}" Margin="20,0,0,0" VerticalOptions="Center" Placeholder="Password"
IsPassword="True" />
        </Frame>

        <Button Text="Sign In" WidthRequest="100" CornerRadius="20" HorizontalOptions="Center" Command="{Binding
LoginCommand}" />

        <StackLayout Orientation="Horizontal" Spacing="5" HorizontalOptions="Center">
            <Label Text="Don't have account?" TextColor="Gray" />
            <Label Text="Sign Up here" TextColor="#50b3f2" />
        </StackLayout>

    </VerticalStackLayout>
</ContentPage>
```

MenuPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="KoalitionAndroidClient.Views.Menu.MenuPage"
             Title="MenuPage">
    <ScrollView>
        <StackLayout Padding="10">
            <Label FontSize="18" Text="Welcome to Koalition" HorizontalOptions="Center"/>

            <Button Text="Create Chat" Command="{Binding EnterCreateGroupChatCommand}" HorizontalOptions="Center" />

            <CollectionView ItemsSource="{Binding GroupChats}" x:Name="groupCollectionView">
                <CollectionView.ItemTemplate>
                    <DataTemplate>
                        <StackLayout Padding="10">
                            <Button
                                Text="{Binding Name}"
                                Command="{Binding BindingContext.EnterChatCommand, Source={x:Reference groupCollectionView}}"
                                CommandParameter="{Binding .}"
                                HorizontalOptions="Center" />
                        </StackLayout>
                    </DataTemplate>
                </CollectionView.ItemTemplate>
```

```
        </CollectionView>

        <CollectionView ItemsSource="{Binding Users}" x:Name="userCollectionView">
          <CollectionView.ItemTemplate>
            <DataTemplate>
              <StackLayout Padding="10">
                <Button
                  Text="{Binding Name}"
                  Command="{Binding BindingContext.EnterPrivateChatCommand, Source={x:Reference
userCollectionView}}"
                  CommandParameter="{Binding .}"
                  HorizontalOptions="Center" />
              </StackLayout>
            </DataTemplate>
          </CollectionView.ItemTemplate>
        </CollectionView>
      </StackLayout>
    </ScrollView>
</ContentPage>
```

PrivateChatPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
         x:Class="KoalitionAndroidClient.Views.Chat.PrivateChatPage"
         Title="PrivateChatPage">
    <Grid>
      <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="Auto" />
      </Grid.RowDefinitions>

      <ScrollView Grid.Row="0">
        <Grid>
          <Grid.RowDefinitions>
            <RowDefinition Height="*" />
          </Grid.RowDefinitions>
          <CollectionView ItemsSource="{Binding GetMessages}" HeightRequest="600" VerticalOptions="FillAndExpand"
x:Name="collectionView">
            <CollectionView.ItemTemplate>
              <DataTemplate>
                <StackLayout Padding="10">
                  <Label Text="{Binding Name}" FontAttributes="Bold" />
                  <Label Text="{Binding Text}" FontSize="Small" TextColor="Gray"/>
                  <Label Text="{Binding Time}" />
                </StackLayout>
              </DataTemplate>
            </CollectionView.ItemTemplate>
          </CollectionView>
        </Grid>
      </ScrollView>

      <StackLayout Grid.Row="1" Padding="10">
        <Entry Placeholder="Enter a message" Text="{Binding MessageText}" />
        <Button Text="Send" Command="{Binding SendMessageCommand}" />
      </StackLayout>
    </Grid>
</ContentPage>
```

GroupChatPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:services="clr-namespace:KoalitionAndroidClient.Services;assembly=KoalitionAndroidClient"
        x:Class="KoalitionAndroidClient.Views.Chat.GroupChatPage"
        Title="GroupChatPage">
  <ContentPage.Resources>
    <ResourceDictionary>
      <services:EditMessageConverter x:Key="EditMessageConverter" />
      <services:SendButtonConverter x:Key="SendButtonConverter" />
    </ResourceDictionary>
  </ContentPage.Resources>

  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="*" />
      <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>

    <ScrollView Grid.Row="0">
      <Grid>
        <Grid.RowDefinitions>
          <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <CollectionView ItemsSource="{Binding Messages}" HeightRequest="600" VerticalOptions="FillAndExpand"
x:Name="collectionView">
          <CollectionView.ItemTemplate>
            <DataTemplate>
              <StackLayout Padding="10">
                <StackLayout.GestureRecognizers>
                  <TapGestureRecognizer Command="{Binding BindingContext.EditMessageCommand,
Source={x:Reference collectionView}}" CommandParameter="{Binding .}" />
                </StackLayout.GestureRecognizers>
                <Label Text="{Binding Name}" FontAttributes="Bold" />
                <Label Text="{Binding Time}" FontSize="Small" TextColor="Gray" />
                <Label Text="{Binding Text}" />
              </StackLayout>
            </DataTemplate>
          </CollectionView.ItemTemplate>
        </CollectionView>
      </Grid>
    </ScrollView>

    <StackLayout Grid.Row="1" Padding="10">
      <Entry Placeholder="Enter a message" Text="{Binding NewMessage}" />
      <Button Text="Send" Command="{Binding SendMessageCommand}" />
    </StackLayout>

  </Grid>
</ContentPage>
```

AddGroupChatPage.xaml

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="KoalitionAndroidClient.Views.AddGroupChatPage.AddGroupChatPage"
```

```
        Title="AddGroupChatPage">
  <VerticalStackLayout
    Spacing="25"
      Padding="30,0"
      VerticalOptions="Center">

    <Label
        Text="Create your chat"
        SemanticProperties.HeadingLevel="Level1"
        FontSize="32"
        HorizontalOptions="Center" />

    <Frame HeightRequest="45" Margin="-20,0,0,0" Padding="0" HasShadow="True" BorderColor="White"
HorizontalOptions="FillAndExpand">
      <Entry Text="{Binding Name}" Margin="20,0,0,0" VerticalOptions="Center" Placeholder="Name" />
    </Frame>

    <Frame HeightRequest="45" Margin="-20,0,0,0" Padding="0" HasShadow="True" BorderColor="White"
HorizontalOptions="FillAndExpand">
      <Entry Text="{Binding Description}" Margin="20,0,0,0" VerticalOptions="Center" Placeholder="Desciption"
IsPassword="True" />
    </Frame>

    <Button Text="Done" WidthRequest="100" CornerRadius="20" HorizontalOptions="Center" Command="{Binding
CreateGroupChatCommand}" />
  </VerticalStackLayout>
</ContentPage>
```

BaseViewModel.cs

```
using Microsoft.Toolkit.Mvvm.ComponentModel;

namespace KoalitionAndroidClient.ViewModels
{
  public partial class BaseViewModel : ObservableObject
  {
    [ObservableProperty]
    private bool _isBusy;

    [ObservableProperty]
    private string _title;
  }
}
```

AppShellViewModel.cs

```
using KoalitionAndroidClient.Views.Startup;
using Microsoft.Toolkit.Mvvm.Input;

namespace KoalitionAndroidClient.ViewModels
{
  public partial class AppShellViewModel : BaseViewModel
  {

    [ICommand]
    async void SignOut()
    {
      if (Preferences.ContainsKey(nameof(App.UserDetails)))
      {
```

```
            Preferences.Remove(nameof(App.UserDetails));
        }
        await Shell.Current.GoToAsync($"//{nameof(LoginPage)}");
    }
  }
}
```

LoginPageViewModel.cs

```
using KoalitionAndroidClient.Models;
using KoalitionAndroidClient.Services;
using KoalitionAndroidClient.Views.Menu;
using Microsoft.Toolkit.Mvvm.ComponentModel;
using Newtonsoft.Json;
using System.Diagnostics;
using System.Windows.Input;

namespace KoalitionAndroidClient.ViewModels.Startup
{
  public partial class LoginPageViewModel : BaseViewModel
  {
    [ObservableProperty]
    private string _login;

    [ObservableProperty]
    private string _password;

    private readonly ILoginService _loginService;
    public LoginPageViewModel(ILoginService loginService)
    {
      _loginService = loginService;
    }
    #region Commands
    private ICommand _loginCommand;
    public ICommand LoginCommand => _loginCommand ??= new Command(Logining);
    async void Logining()
    {
      Debug.WriteLine("Logining method started");

      if (!string.IsNullOrWhiteSpace(Login) && !string.IsNullOrWhiteSpace(Password))
      {
        Debug.WriteLine("Login and password are not null or whitespace");

        // calling api
        var response = await _loginService.Authenticate(new LoginRequest
        {
          Login = Login,
          Password = Password
        });

        if (response != null)
        {
          Debug.WriteLine("Authentication succeeded");

          response.UserDetails.Login = Login;

          if (Preferences.ContainsKey(nameof(App.UserDetails)))
          {
            Preferences.Remove(nameof(App.UserDetails));
          }
```

```
                string userDetailStr = JsonConvert.SerializeObject(response.UserDetails);
                Preferences.Set(nameof(App.UserDetails), userDetailStr);
                App.UserDetails = response.UserDetails;
                App.Token = response.Token;
                await Shell.Current.GoToAsync($"//{nameof(MenuPage)}");
            }
            else
            {
                Debug.WriteLine("Authentication failed");

                await AppShell.Current.DisplayAlert("Invalid Login Or Password", "Invalid Login or Password", "OK");
            }
        }
        else
        {
            Debug.WriteLine("Login or password are null or whitespace");
        }
    }
    #endregion
  }
}
```

MenuPageViewModel.cs

```
using KoalitionAndroidClient.Controls;
using KoalitionAndroidClient.Helpers;
using KoalitionAndroidClient.Models;
using KoalitionAndroidClient.Services;
using KoalitionAndroidClient.ViewModels.Chat;
using KoalitionAndroidClient.Views.Chat;
using Newtonsoft.Json;
using System.Collections.ObjectModel;
using System.Net.Http.Headers;
using System.Windows.Input;

namespace KoalitionAndroidClient.ViewModels.Menu
{
  public class MenuPageViewModel : BaseViewModel
  {
    public ObservableCollection<GroupChatResponse> GroupChats { get; set; } =
      new ObservableCollection<GroupChatResponse>();
    private ObservableCollection<UserBasicInfo> _users;
    public ObservableCollection<UserBasicInfo> Users
    {
      get { return _users; }
      set
      {
        _users = value;
        OnPropertyChanged(nameof(Users));
      }
    }

    public readonly ILoginService _loginService;
    private ObservableCollection<ChatMessageResponse> _messages;
    public GroupChatResponse _selectedGroupChat;

    public ObservableCollection<ChatMessageResponse> Messages
    {
      get { return _messages; }
      set
      {
```

```
            _messages = value;
            OnPropertyChanged(nameof(Messages));
        }
    }
    public ICommand EnterChatCommand => new Command(EnterChat);
    public ICommand EnterCreateGroupChatCommand => new Command(EnterCreateGroupChat);
    public ICommand EnterPrivateChatCommand => new Command(EnterPrivateChat);
    public MenuPageViewModel(ILoginService loginService)
    {
        _loginService = loginService;
        AppShell.Current.FlyoutHeader = new FlyoutHeaderControl();
        GetUsers();
    }
    public async void EnterPrivateChat(object selectedUser)
    {
        var user = selectedUser as UserBasicInfo;

        var viewModel = new PrivateChatPageViewModel(user);

        var privateChatPage = new PrivateChatPage(viewModel);
        await Shell.Current.Navigation.PushAsync(privateChatPage);
    }

    public async void EnterChat(object groupChat)
    {
        var selectedGroupChat = groupChat as GroupChatResponse;

        var viewModel = new GroupChatPageViewModel(selectedGroupChat);
        await viewModel.GetUsersAndMessages();

        var groupChatPage = new GroupChatPage(viewModel);
        await Shell.Current.Navigation.PushAsync(groupChatPage);
    }

    public void GetGroupChats()
    {
        Task.Run(async () =>
        {
            var groupChatList = await _loginService.GetGroupChats();

            App.Current.Dispatcher.Dispatch(() =>
            {
                if (groupChatList?.Count > 0)
                {
                    foreach (var groupChat in groupChatList)
                    {
                        GroupChats.Add(groupChat);
                    }
                }
            });
        });
    }

    public async void EnterCreateGroupChat()
    {
        await Shell.Current.GoToAsync("AddGroupChatPage");
    }

    private async Task GetUsers()
    {
        using HttpClient httpClient = new HttpClient();
        httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", App.Token);
```

```
        var response = await httpClient.GetAsync($"{ApiPlatformUrlHelper.GetPlatformApiUrl()}/api/Users/allUsers");

        if (response.IsSuccessStatusCode)
        {
            var content = await response.Content.ReadAsStringAsync();
            var users = JsonConvert.DeserializeObject<List<UserBasicInfo>>(content);
            Users = new ObservableCollection<UserBasicInfo>(users);
        }
        else
        {
        }
    }

  }
}
```

PrivateChatPageViewModel.cs

```
using KoalitionAndroidClient.Helpers;
using KoalitionAndroidClient.Models;
using Newtonsoft.Json;
using System.Collections.ObjectModel;
using System.Net.Http.Headers;
using System.Text;
using System.Windows.Input;

namespace KoalitionAndroidClient.ViewModels.Chat;

public class PrivateChatPageViewModel : BaseViewModel
{
  private int _recipientId;
  private string _messageText;
  private ObservableCollection<SendMessageRequest> _messages;
  private UserBasicInfo _selectedUser;
  private ObservableCollection<ChatMessageResponse> _getMessages;

  public UserBasicInfo SelectedUser { get; set; }

  public ObservableCollection<SendMessageRequest> Messages
  {
    get { return _messages; }
    set
    {
      _messages = value;
      OnPropertyChanged(nameof(Messages));
    }
  }
  public ObservableCollection<ChatMessageResponse> GetMessages
  {
    get { return _getMessages; }
    set
    {
      _getMessages = value;
      OnPropertyChanged(nameof(GetMessages));
    }
  }

  public string MessageText
  {
    get { return _messageText; }
```

```
        set
        {
            _messageText = value;
            OnPropertyChanged(nameof(MessageText));
        }
    }

    public ICommand SendMessageCommand { get; }

    public PrivateChatPageViewModel(UserBasicInfo user)
    {
        //_recipientId = recipientId;
        SendMessageCommand = new Command(async () => await SendMessage());
        _selectedUser = user;
        LoadMessages();

    }

    public async Task SendMessage()
    {
        using HttpClient httpClient = new HttpClient();
        httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", App.Token);

        var message = new SendMessageRequest
        {
            Text = MessageText,
            Id = _selectedUser.UserId,
        };
        var messageJson = JsonConvert.SerializeObject(message);
        var content = new StringContent(messageJson, Encoding.UTF8, "application/json");
        var response = await
httpClient.PostAsync($"{ApiPlatformUrlHelper.GetPlatformApiUrl()}/api/privatechats/sendMessage?recipientId={_selectedU
ser.UserId}", content);
        if (response != null)
        {
            MessageText = string.Empty;
            await LoadMessages();
        }
    }

    public async Task LoadMessages()
    {
        using HttpClient httpClient = new HttpClient();
        httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", App.Token);

        var response = await
httpClient.GetAsync($"{ApiPlatformUrlHelper.GetPlatformApiUrl()}/api/privatechats?recipientId={_selectedUser.UserId}");
        var privateChatJson = await response.Content.ReadAsStringAsync();
        var usersAndMessages = JsonConvert.DeserializeObject<List<ChatMessageResponse>>(privateChatJson);

        foreach (var message in usersAndMessages)
        {
            var sender = usersAndMessages.FirstOrDefault(u => u.UserId == message.UserId);
            if (sender != null)
            {
                message.Name = sender.Name;
            }
        }
        GetMessages = new ObservableCollection<ChatMessageResponse>(usersAndMessages);
    }
}
```

GroupChatPageViewModel.cs

```
using KoalitionAndroidClient.Helpers;
using KoalitionAndroidClient.Models;
using Newtonsoft.Json;
using System.Collections.ObjectModel;
using System.Net.Http.Headers;
using System.Text;
using System.Windows.Input;

namespace KoalitionAndroidClient.ViewModels.Chat
{
  public class GroupChatPageViewModel : BaseViewModel
  {
    private ObservableCollection<ChatMessageResponse> _messages;
    private ObservableCollection<SendMessageRequest> _sendMessageText;
    private ObservableCollection<UserBasicInfo> _users;
    private GroupChatResponse _selectedGroupChat;
    public GroupChatResponse SelectedGroupChat { get; set; }


    public ObservableCollection<ChatMessageResponse> Messages
    {
      get { return _messages; }
      set
      {
        _messages = value;
        OnPropertyChanged(nameof(Messages));
      }
    }

    public ObservableCollection<UserBasicInfo> User
    {
      get { return _users; }
      set
      {
        _users = value;
        OnPropertyChanged(nameof(User));
      }
    }

    public ObservableCollection<SendMessageRequest> SendMessageRequests
    {
      get => _sendMessageText;
      set
      {
        _sendMessageText = value;
        OnPropertyChanged();
      }
    }

    private string _newMessage;
    public string NewMessage
    {
      get { return _newMessage; }
      set
      {
        _newMessage = value;
        OnPropertyChanged(nameof(NewMessage));
      }
    }
```

```
private string _editMessage;
public string EditMessageText
{
    get { return _editMessage; }
    set
    {
        _editMessage = value;
        OnPropertyChanged(nameof(EditMessageText));
        IsEditing = !string.IsNullOrEmpty(_editMessage);
    }
}

public ObservableCollection<UserBasicInfo> Users
{
    get { return _users; }
    set
    {
        _users = value;
        OnPropertyChanged(nameof(Users));
    }
}

private bool _isEditing;
public bool IsEditing
{
    get { return _isEditing; }
    set
    {
        _isEditing = value;
        OnPropertyChanged(nameof(IsEditing));
        OnPropertyChanged(nameof(EditMessageText));
    }
}

private ChatMessageResponse _selectedMessage;
public ChatMessageResponse SelectedMessage
{
    get { return _selectedMessage; }
    set
    {
        _selectedMessage = value;
        OnPropertyChanged(nameof(SelectedMessage));
        IsEditing = (_selectedMessage != null);
    }
}

public ICommand SendMessageCommand { get;set; }
public ICommand EditMessageCommand { get; }
public GroupChatPageViewModel(GroupChatResponse selectedGroupChat)
{
    //EditMessageCommand = new Command(async () => await EditSelectedMessage());
    EditMessageCommand = new Command<ChatMessageResponse>(EditMessage);
    SendMessageCommand = new Command(async () =>
    {
        if (IsEditing)
        {
            // Edit the selected message
            await EditSelectedMessage();
        }
        else
        {
            // Send a new message
```

```
            await SendMessage();
        }
    });

    _selectedGroupChat = selectedGroupChat;
    GetUsersAndMessages();
}


public void EditMessage(ChatMessageResponse message)
{
    EditMessageText = message.Text;
    SelectedMessage = message;
    SelectedMessage.MessageId = message.MessageId;
    IsEditing = true;
}

public async Task EditSelectedMessage()
{
    using HttpClient httpClient = new HttpClient();
    httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", App.Token);

    var GroupChatId = _selectedGroupChat.Id;
    var MessageId = SelectedMessage.MessageId;
      //Text = EditMessageText,

    var text = EditMessageText;

    var message = new SendMessageRequest { Text = text };
    var messageJson = JsonConvert.SerializeObject(message);
    var content = new StringContent(messageJson, Encoding.UTF8, "application/json");
    var response = await
httpClient.PutAsync($"http://10.0.2.2:5127/api/groupchats/{_selectedGroupChat.Id}/messages/{SelectedMessage.MessageI
d}", content);

    if (response.IsSuccessStatusCode)
    {
      // Clear the EditMessageText property
      EditMessageText = string.Empty;
      SelectedMessage = null;
      IsEditing = false;
      await GetUsersAndMessages();
    }
}



public async Task GetUsersAndMessages()
{
    using HttpClient httpClient = new HttpClient();
    httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", App.Token);

    // Get all users
    var usersResponse = await httpClient.GetAsync($"{ApiPlatformUrlHelper.GetPlatformApiUrl()}/api/Users/allUsers");
    var usersJson = await usersResponse.Content.ReadAsStringAsync();
    var users = JsonConvert.DeserializeObject<List<UserBasicInfo>>(usersJson);

    var messagesResponse = await
httpClient.GetAsync($"{ApiPlatformUrlHelper.GetPlatformApiUrl()}/api/groupchats/{_selectedGroupChat.Id}/messages");
    var messagesJson = await messagesResponse.Content.ReadAsStringAsync();
    var messages = JsonConvert.DeserializeObject<List<ChatMessageResponse>>(messagesJson);
```

```
            foreach (var message in messages)
            {
                var sender = users.FirstOrDefault(u => u.UserId == message.UserId);
                if (sender != null)
                {
                    message.Name = sender.Name;
                }
            }
            Messages = new ObservableCollection<ChatMessageResponse>(messages);
        }


        public async Task SendMessage()
        {
            using HttpClient httpClient = new HttpClient();
            httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", App.Token);
            var message = new SendMessageRequest
            {
                Text = NewMessage,
                Id = _selectedGroupChat.Id
            };
            var messageJson = JsonConvert.SerializeObject(message);
            var content = new StringContent(messageJson, Encoding.UTF8, "application/json");
            var response = await
httpClient.PostAsync($"{ApiPlatformUrlHelper.GetPlatformApiUrl()}/api/groupchats/{_selectedGroupChat.Id}/messages/sen
dMessage", content);
            if (response.IsSuccessStatusCode)
            {
                NewMessage = string.Empty;
                await GetUsersAndMessages();
            }

        }


    }
}
```

AddGroupChatPageViewModel.cs

```
using KoalitionAndroidClient.Helpers;
using KoalitionAndroidClient.Models;
using KoalitionAndroidClient.ViewModels.Menu;
using KoalitionAndroidClient.Views.Menu;
using Newtonsoft.Json;
using System.Collections.ObjectModel;
using System.Net.Http.Headers;
using System.Text;
using System.Windows.Input;

namespace KoalitionAndroidClient.ViewModels.AddGroupChatPageViewModel
{
    public class AddGroupChatPageViewModel:BaseViewModel
    {
        private ObservableCollection<CreateGroupChatRequest> _groupChat;
        public ObservableCollection<CreateGroupChatRequest> GroupChat
        {
            get  => _groupChat;
            set
            {
```

```
            _groupChat = value;
            OnPropertyChanged();
        }
    }

    private string _name;
    private string _description;

    public string Name
    {
        get { return _name; }
        set
        {
            _name = value;
            OnPropertyChanged(nameof(Name));
        }
    }

    public string Description
    {
        get { return _description; }
        set
        {
            _description = value;
            OnPropertyChanged(nameof(Description));
        }
    }


    public ICommand CreateGroupChatCommand { get; set; }
    public AddGroupChatPageViewModel()
    {
        CreateGroupChatCommand = new Command<object>(async (param) => await CreateGroupChat());
    }

    public async Task CreateGroupChat()
    {
        using var client = new HttpClient();
        client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", App.Token);
        var groupChat = new CreateGroupChatRequest
        {
            Name = Name,
            Description = Description
        };

        var groupChatJson = JsonConvert.SerializeObject(groupChat);
        var content = new StringContent(groupChatJson, Encoding.UTF8, "application/json");
        var response = await
client.PostAsync($"{ApiPlatformUrlHelper.GetPlatformApiUrl()}/api/GroupChat/createGroupChat", content);

        await Shell.Current.GoToAsync("///MenuPage");
        if (Application.Current.MainPage is MenuPage menuPage && menuPage.BindingContext is MenuPageViewModel
menuViewModel)
        {
            menuViewModel.GetGroupChats();
            await Application.Current.MainPage.Navigation.PushAsync(menuPage);
        }
        else
        {
            Console.WriteLine("Error: Unable to cast MainPage to MenuPage or BindingContext to MenuPageViewModel.");
        }
    }
```

```
    }
}
```

LoginService.cs

```
using KoalitionAndroidClient.Helpers;
using KoalitionAndroidClient.Models;
using Newtonsoft.Json;
using System.Text;

namespace KoalitionAndroidClient.Services
{
    public class LoginService : ILoginService
    {

        public async Task<LoginResponse> Authenticate(LoginRequest loginRequest)
        {
            using (var client = new HttpClient())
            {
                string loginRequestStr = JsonConvert.SerializeObject(loginRequest);

                var response = await client.PostAsync($"{ApiPlatformUrlHelper.GetPlatformApiUrl()}/api/Users/Login",
                    new StringContent(loginRequestStr, Encoding.UTF8, "application/json"));

                if (response != null && response.StatusCode == System.Net.HttpStatusCode.OK)
                {
                    var json = await response.Content.ReadAsStringAsync();
                    return JsonConvert.DeserializeObject<LoginResponse>(json);
                }
                else
                {
                    return null;
                }
            }
        }

        public async Task<List<GroupChatResponse>> GetGroupChats()
        {
            using (var client = new HttpClient())
            {
                client.DefaultRequestHeaders.Add("Authorization", "Bearer " + App.Token);
                var response = await client.GetAsync($"{ApiPlatformUrlHelper.GetPlatformApiUrl()}/api/GroupChat/getChats");

                if (response.StatusCode == System.Net.HttpStatusCode.OK)
                {
                    var json = await response.Content.ReadAsStringAsync();
                    return JsonConvert.DeserializeObject<List<GroupChatResponse>>(json);
                }
                else
                {
                    return null;
                }
            }
        }
    }
}
```

UserBasicInfo.cs

```
namespace KoalitionAndroidClient.Models
{
   public class UserBasicInfo
   {
      public int UserId { get; set; }
      public string Login { get; set; }
      public string Name { get; set; }
      public string Email { get; set; }
      public string Password { get; set; }
   }
}
```

SendMessageResponse.cs

```
namespace KoalitionAndroidClient.Models
{
   class SendMessageResponse
   {
      public string Text { get; set; }
      public DateTime Time { get; set; }
   }
}
```

SendMessageRequest.cs

```
namespace KoalitionAndroidClient.Models
{
   public class SendMessageRequest
   {
      public int Id { get; set; }
      public string Text { get; set; }
   }
}
```

SelectedUserRequest.cs

```
using Newtonsoft.Json;

namespace KoalitionAndroidClient.Models
{
   [JsonObject]
   public class SelectedUserRequest
   {
      public int Id { get; set; }
      public string Name { get; set; }
   }
}
```

LoginResponse.cs

```
namespace KoalitionAndroidClient.Models
{
   public class LoginResponse
   {
      public string Token { get; set; }
      public UserBasicInfo UserDetails { get; set; }
   }
```

```
}
```

LoginRequest.cs

```
namespace KoalitionAndroidClient.Models
{
  public class LoginRequest
  {
    public string Login { get; set; }
    public string Password { get; set; }
  }
}
```

GroupChatResponse.cs

```
using Newtonsoft.Json;

namespace KoalitionAndroidClient.Models
{
  [JsonObject]
  public class GroupChatResponse
  {
    public int Id { get; set; }
    public string Name { get; set; }
  }
}
```

EditMessageRequest.cs

```
namespace KoalitionAndroidClient.Models
{
  public class EditMessageRequest
  {
    public int GroupChatId { get; set; }
    public int MessageId { get; set; }
    public string Text { get; set; }
  }
}
```

CreateGroupChatRequest.cs

```
namespace KoalitionAndroidClient.Models
{
  public class CreateGroupChatRequest
  {
    public int Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
  }
}
```

ChatMessageResponse.cs

```
using Newtonsoft.Json;
```

```
namespace KoalitionAndroidClient.Models
{
   public class ChatMessageResponse
   {
      [JsonProperty("messageId")]
      public int MessageId { get; set; }
      public string Text { get; set; }
      public DateTime Time { get; set; }
      public int UserId { get; set; }
      public string Name { get; set; }
   }
}
```

ApiPlatformUrlHelper.cs

```
namespace KoalitionAndroidClient.Helpers
{
   public static class ApiPlatformUrlHelper
   {
      public static string GetPlatformApiUrl()
      {
#if __ANDROID__
      return "http://10.0.2.2:5127";
#else
      return "http://localhost:5127";
#endif
      }
   }
}
```

FlyoutHeaderControl.cs

```
<?xml version="1.0" encoding="utf-8" ?>
<StackLayout Padding="20" xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
         xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
         x:Class="KoalitionAndroidClient.Controls.FlyoutHeaderControl">
   <StackLayout Orientation="Horizontal" Spacing="10">
      <Label x:Name="lblUserName"  FontSize="28" FontAttributes="Bold" />
   </StackLayout>
   <Label x:Name="lblUserEmail" FontSize="18" />
</StackLayout>
```

ДОДАТОК Б

ВІДЗИВ

на дипломний проект бакалавра
на тему:
«Розробка веб-додатку для візуалізації планування задач»

студента групи 122-16-2 Мухтаряна Максима Артуровича

ДОДАТОК В

РЕЦЕНЗІЯ

на дипломний проект бакалавра
на тему:
«Розробка кросплатформного додатку для організації чату з використанням .NET та мови C#»
студента групи 122-20-ск1 Ілюшина Олександра Сергійовича

ДОДАТОК Г

Відзив керівника економічного розділу

ДОДАТОК Д

ПЕРЕЛІК ФАЙЛІВ НА ОПТИЧНОМУ НОСІЇ

Ім'я файла	**Опис**
Пояснювальні документи	
Диплом.doc	Пояснювальна записка до дипломного проекту. Документ Word.
Диплом.pdf	Пояснювальна записка до дипломного проекту в форматі PDF
Програма	
Koalition.zip	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація.ppt	Презентація дипломного проекту