МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

Чат для мобільних пристроїв з використанням технології Xamarin

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма **«Прикладне програмування»**

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи ПП-42
_____Компанець І.М_____
(прізвище та ініціали)

Керівник _____Ващіліна О.В_____
(прізвище та ініціали)
_____к.ф. - м.н., доцент_____
(науковий ступінь, звання)

Унікальність тексту 90,8 %

Випускна кваліфікаційна робота бакалавра допущена до захисту
Рішенням кафедри *прикладних інформаційних систем*
Протокол № 14 від 23.05.2023р.
зав. кафедри _____ Плескач В. Л.

Київ – 2023

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

НАЗВА ТЕМИ: «Чат для мобільних пристроїв з використанням технології Xamarin»

Освітня програма: Прикладне програмування
Спеціальність: Комп'ютерні науки

ПІБ	Підпис
Компанець Ілля Михайлович	

ТЕМА РОБОТИ

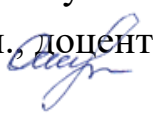
Чат для мобільних пристроїв з використання технології Xamarin Mobile chat application using Xamarin technology

МЕТА БАКАЛАВРСЬКОЇ РОБОТИ, ЗАВДАННЯ

Мета бакалаврської роботи: забезпечення ефективної комунікації у формі чату для мобільних пристроїв з використанням технології Xamarin

План роботи:

- Дослідження сучасних засобів та підходів до спілкування через мобільні пристрої
- Здійснити аналіз програмно-технологічних рішень для розробки мобільного застосунку чату з використанням технології Xamarin
- Проектування та програмна реалізація мобільного застосунка чату з використанням технології Xamarin

ПІБ, ступінь, звання наукового керівника роботи: Ващіліна Олена Валеріївна, к.ф.-м.н., доцент 

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

№з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	14.10.2022	Виконано
2.	Видача завдання кваліфікаційної роботи бакалавра	24.10.2022	Виконано
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	31.10.2022	Виконано
4.	Затвердження плану кваліфікаційної роботи бакалавра	01.11.2022	Виконано
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	08.11.2022	Виконано
6.	Підготовка і подання науковому керівнику першого варіанту I розділу роботи	21.12.2022	Виконано
7.	Підготовка і подання науковому керівнику першого варіанту II розділу роботи	31.01.2023	Виконано
8.	Підготовка і подання науковому керівнику першого варіанту III розділу роботи	30.03.2023	Виконано
9.	Подання роботи у першому варіанті	28.04.2023	Виконано
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	03.05.2023	Виконано
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	**22.05.2023**	Виконано
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на	26.05.2023	Виконано

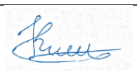
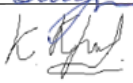
	кафедру		
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	12.06.2023	Виконано
14.	Захист кваліфікаційної роботи бакалавра	28.06.2023	Виконано

Здобувач вищої освіти _____
(підпис)

Керівник _____
(підпис)

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Складові частини дипломної роботи	Обсяг, арк.
Титульний аркуш	1
Завдання до дипломної роботи	1
Календарний план дипломної роботи	1
Відомість дипломної роботи	1
Анотація	1
Анотація (іноземною мовою - англійською)	1
Зміст	1
Перелік скорочень, умовних позначень, термінів	1
Вступ	2
1	28
2	17
3	12
Висновки	2
Перелік використаних джерел	4
Додатки	3

				ДП ХХХХ 00.000.00		
	ПІБ	Підп.	Дата			
Розробн.	Компанець І.М.			Відомість дипломної роботи	Лист	Листів
Керівн.	Ващіліна О.В.					78
Н/контр.	Кравченко К.В.					
Зав. каф.	Плескач В.Л.					

АНОТАЦІЯ

Дипломна робота: 73 с., 29 рис., 31 джерело, 3 дод.

Метою роботи є забезпечення ефективної комунікації у формі чату для мобільних пристроїв з використанням технології Xamarin

Для досягнення мети роботи потрібно вирішити такі **завдання**:

- ·Дослідити сучасні засоби та підходи до спілкування через мобільні пристрої;
- ·Здійснити аналіз програмно-технологічних рішень для розробки мобільного застосунку чату з використанням технології Xamarin;
- ·Проектування та програмна реалізація мобільного застосунка чату з використанням технології Xamarin

Об'єктом дослідження є процеси забезпечення комунікації між користувачами мобільних пристроїв

Предметом дослідження є програмно-технічні засади та концепції розробки програмних продуктів для мобільних пристроїв з використанням технології Xamarin .

Методи дослідження: метод аналізу для опрацювання існуючих рішень у сфері мобільних застосунків-чатів; метод наукового моделювання для створення діаграм та моделей; описовий метод був застосований при формуванні алгоритму розробки та кроків моделювання проекту, опису використаних технологій, опису можливостей застосунку; метод синтезу для вивчення необхідних технології, архітектурних рішень, інших програмних інструментів для виконання бакалаврської роботи

Ключові слова: застосунок-чат, Xamarin, кросплатформеність

ABSTRACT

Thesis: 73 p., 29 fig., 31 ref., 3 app.

The purpose of this thesis enabling effective communication in the form of a chat for mobile devices using Xamarin technology.

To achieve the goal of the project, the following **tasks** need to be accomplished:

– Research modern tools and approaches for communication through mobile devices;

– Analyze software and technological solutions for developing a mobile chat application using Xamarin technology;

– Design and implement a mobile chat application using Xamarin technology.

Research Object

The widespread use of chat applications for mobile devices and their popularity among users.

Research Subject

Software and technical foundations and concepts for developing communication applications for users.

Research Methods

Study of existing software and technical solutions in the field of internet communication through chat applications. Description of architecture construction approaches and chat application development. Modeling during the creation of diagrams for the developed software application. Development of a mobile chat application using Xamarin technology.

Keywords: chat application, Xamarin, cross-platform

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

BaaS – Backend as a Service – серверна частина як сервіс.

API – Application Programming Interface – інтерфейс програмування застосунків.

P2P – Peep to Peer – архітектура мереж, в якій кожен вузол може бути і клієнтом, і сервером.

FCM – Firebase Cloud Messaging.

ВСТУП

У наш час спілкування стає все більш цифровим та мобільним. Бути на зв’язку стало необхідним для людей у всіх сферах життя, включаючи особисте спілкування, бізнес та соціальні мережі. Для вирішення цієї проблеми більшість людей обирають для себе щоденне використання мобільних застосунків, у тому числі й чатів, тому-що такі застосунки надають багато можливостей для прийняття рішень, постійний доступ до отримання та надсилання інформації незалежно від місця та часу, що робить їх незамінними для сучасного способу життя.

Актуальність дослідження. У сучасному світі, переповненому інформацією, кожній людині потрібно весь час «тримати руку на пульсі подій», де мобільність та швидкість є важливими, застосунки-чати стають зручним інструментом для спілкування, покращуючи продуктивність та забезпечуючи постійний доступ до важливих особистих та професійних контактів. Розробка застосунку-чату є доцільною, тому-що вони використовуються у різних сферах, включаючи особисте спілкування, бізнес та комунікацію в соціальних мережах.

Метою дипломної роботи є забезпечення ефективної комунікації у формі чату для мобільних пристроїв з використанням технології Xamarin

Завдання дослідження:

– проаналізувати сучасні підходи для розробки мобільних застосунків та їх подальшого впровадження;

– дослідити можливості технології Xamarin для розробки програмних продуктів;

– проаналізувати наявні сучасні продукти в сфері застосунківків для спілкування для формування вимог до реалізації;

– спроектувати та розробити застосунок.

Об’єктом дослідження є процеси забезпечення комунікації між користувачами мобільних пристроїв

Предметом дослідження є програмно-технічні засади та концепції розробки програмних продуктів для мобільних пристроїв з використанням технології Xamarin .

Методи дослідження Для виконання бакалаврської роботи, розкриття теми та формування звіту про розробку програмного продукту були використані такі методи:

- метод аналізу застосовано при опрацюванні існуючих рішень у сфері мобільних застосунків-чатів;
- метод наукового моделювання був використаний для створення діаграм та моделей, які необхідні для візуалізації потрібних елементів бакалаврської роботи;
- описовий метод був застосований при формуванні алгоритму розробки та кроків моделювання проекту, опису використаних технологій для розробки, а також для опису можливостей застосунку;
- метод синтезу для вивчення необхідних технології, архітектурних рішень, інших програмних інструментів для виконання бакалаврської роботи

Практичне значення одержаних результатів полягає у тому, що розроблений програмний продукт – «застосунок-чат» може стати корисним та зручним рішенням для щоденного використання людей у всіх сферах сучасного життя для того щоб підвищити їхню ефективність у повсякденному житті завдяки

доступу до необхідних контактів та обміну інформацією з ними у будь-який час та за будь-яких обставин.

Структура роботи: Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, розподілених на підрозділі та висновку.

РОЗДІЛ 1

ДОСЛІДЖЕННЯ СУЧАСНИХ ЗАСОБІВ ТА ПІДХОДІВ ДО МОБІЛЬНОГО СПІЛКУВАННЯ

1.1 Огляд та причини популярності мобільних застосунків-чатів.

Напевно, нікому не треба доводити факт того, що застосунки для спілкування міцно увійшли у наше життя. Вони забезпечили нову свободу спілкування, став більш гнучкою альтернативою наземного та стільникового зв’язку. Зі зростанням популярності мобільних пристроїв застосунки для спілкування «трансформувались» (з’явилася підтримка екранів невеликої чіткості та процесорів архітектури ARM) та «розмножились», що цілком закономірно призвело до загострення конкуренції у сегменті[1]. На цей момент у каталогах Google Play та App Store присутні декілька десятків застосунків-месенджерів, кожен з яких може претендувати на звання кращого. На мій погляд, є кілька основних взаємопов’язаних причин популярності мобільних інтернет-месенджерів:

- мобільні пристрої стають дешевше з кожним роком, при цьому зростає їхня обчислювальна потужність;
- користувачі з усього світу «мігрують» до мобільних мереж, де активно споживають трафік;
- трафік мобільних мереж переходить до мобільних інтернет месенджерів.

Розглянемо кожну з них.

Причина #1. Мобільні пристрої стають дешевшими, їх обчислювальна потужність росте.

За останні 15 років вартість смартфонів помітно знизилась, а їх характеристики, навпаки, покращились. Наприклад, випущений у 2007 році флагманський смартфон Sony Ericcson P1i мав ціну у 650 євро, при цьому він оснащувався 128 МБ оперативної пам’яті (з них лише 79 залишаються вільними після запуску) та одноядерним процессором Philips Nexperia PNX4008 з максимальною частотою 220 МГЦ. У 2013 році з'явився на ринку смартфон Sony Xperia ZL за ціною 550 євро, який має 2 ГБ оперативної пам'яті та чотирьохядерний процесор Qualcomm APQ8064 з тактовою частотою 1,5 ГГц. Нині, за ту ж саму ціну у перерахунку на національну валюту України – гривню(навіть з урахуванням інфляції), ми можемо придбати Samsung Galaxy A54 з тактовою частотою 2,4 ГГц, 8 ядрами, 128 ГБ внутрішньої пам’яті та 6 ГБ оперативної пам’яті. – помітно кращі характеристики попри те що це навіть не флагманська модель Samsung.

Таблиця 1.1 - Порівняння характеристик мобільних пристроїв

Назва моделі	Sony Ericcson P1i	Sony Xperia ZL	Samsung Galaxy A54
Рік випуску	2007	2013	2023
Флагманська модель	так	так	ні
Частота процесора	220 МГЦ	1,5 ГГЦ	2,4 ГГЦ
Оперативна пам’ять	128 МБ	2 ГБ	6 ГБ
Внутрішня	256 МБ	16 ГБ	128 ГБ

пам’ять			
Кількість ядер	1	4	8
Ціна	650 євро	550 євро	20 000 грн

Рисунок 1.1 - Sony Ericsson P1i A54 Рисунок 1.2 - Samsung Galaxy

Зростання обчислювальних можливостей мобільних пристроїв, а також перехід на більш сучасні технології передачі даних (LTE) з одного боку сприяє споживанню трафіку в мобільних мережах (включаючи інтернет-месенджери), з іншого боку значно спрощує розробку мобільних застосунків. Якщо раніше при створенні застосунку обов'язково доводилося враховувати невеликі обчислювальні можливості процесора та невеликий обсяг ОЗУ, то тепер ці проблеми відступили на другий план. Нині потужні смартфони доступні за більш помірну ціну.

Емпіричні спостереження про постійне зниження цін на смартфони підтверджуються даними аналітичних агентств. За підрахунками IDC, середня ціна смартфона у 2011 році становила $448, у 2012 році вона вже склала $407. У 2021 році середня ціна на смартфон досягла позначки 317 доларів [2][3].

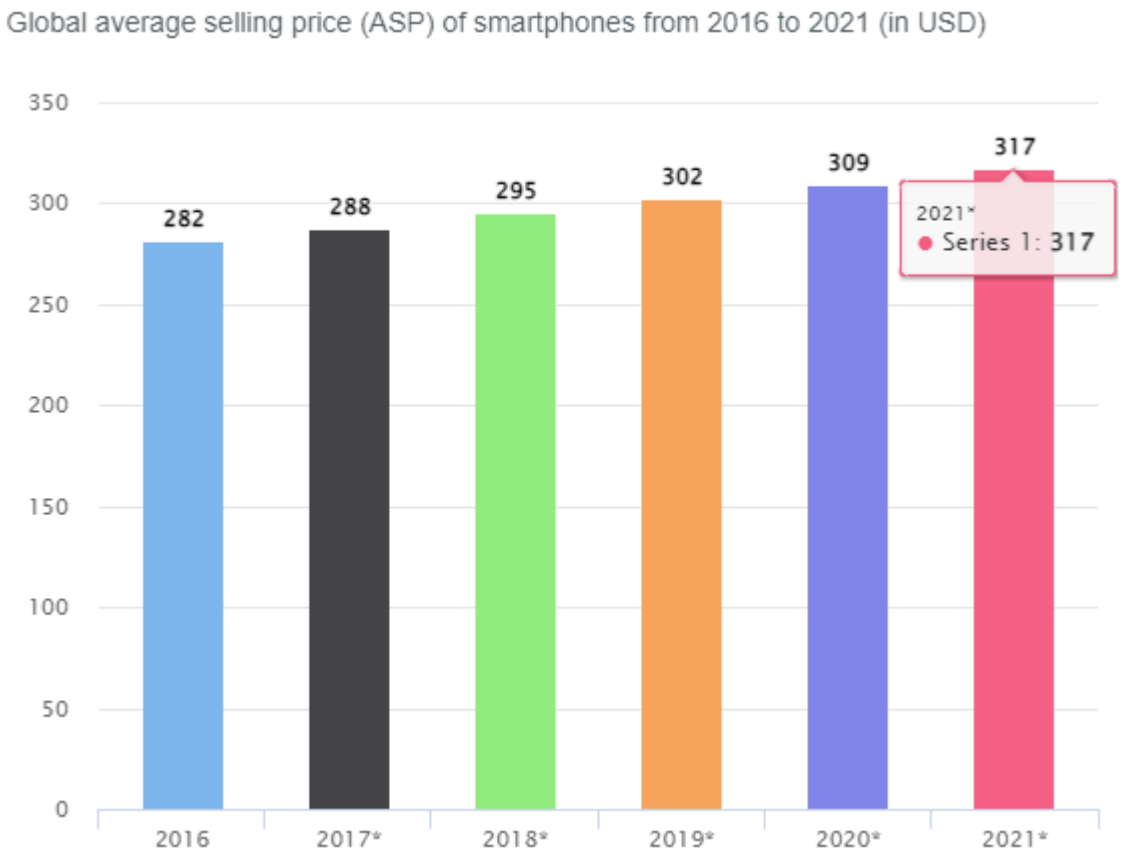


Рисунок 1.3 – Середня ціна на смартфон з 2016 по 2021 рік

За графіком можна помітити що вона зростає, але причина цьому – світова дефляція долару та світові події. Наглядно видно, що ціна знизилась на 100 доларів, а потужність навіть не флагманських моделей суттєво зросла.

Причина #2: Міграція інтернет-трафіку до мобільних мереж та зростання кількості користувачів мобільного доступу до мережі Інтернет.

На початку другого кварталу 2023 року загальна кількість користувачів Інтернету по всьому світу склала 5,18 мільярда осіб, що еквівалентно 64,6

відсоткам загальної світової популяції. Кількість користувачів Інтернету продовжує зростати, і останні дані свідчать про збільшення світової підключеної популяції на практично 147 мільйонів користувачів протягом 12 місяців до квітня 2023 року. Зростання на рік до року, яке складає майже 3 відсотки, є значно повільнішим, ніж зростання, яке спостерігалося наприкінці попереднього десятиліття. Проте, регулярні затримки у обробці та публікації досліджень щодо прийняття Інтернету, ймовірно, означають, що реальний темп зростання вищий, ніж ці останні показники свідчать[4].

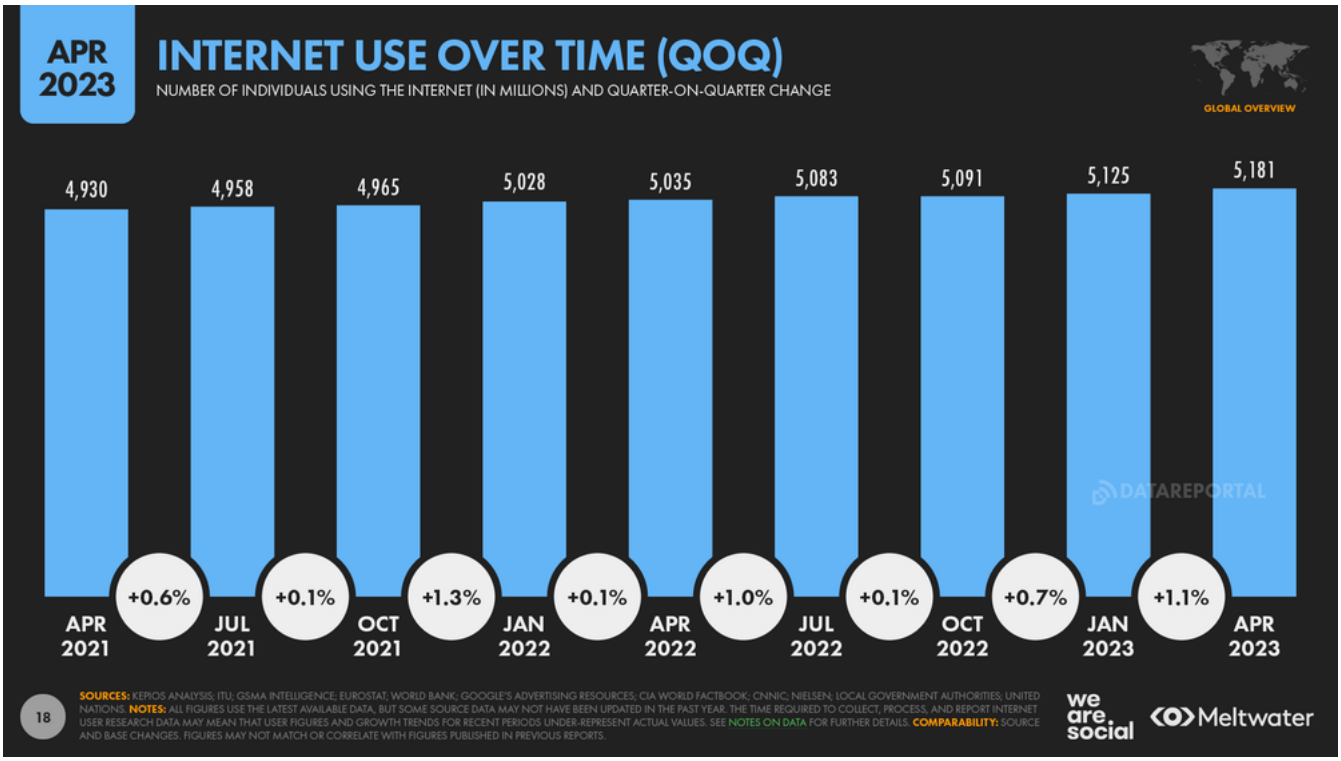


Рисунок 1.4 – динаміка зміни кількості користувачів інтернетом помісячно за 2 роки

Через зростання глобального доступу до Інтернету кількість людей, які не мають доступу до мережі зменшилася до 2,85 млрд, більшість з яких проживає у південно-східній Азії та Африці. Проте, поточні тенденції свідчать що до кінця 2023 року Інтернетом зможуть користуватися дві третини населення світу.

Переважна більшість користувачів інтернету в світі – 95% - використовують мобільний телефон для доступу в інтернет, принаймні частково, а мобільні телефони займають більше 57 відсотків нашого часу онлайн, а також майже 60% веб-трафіку світу[4].

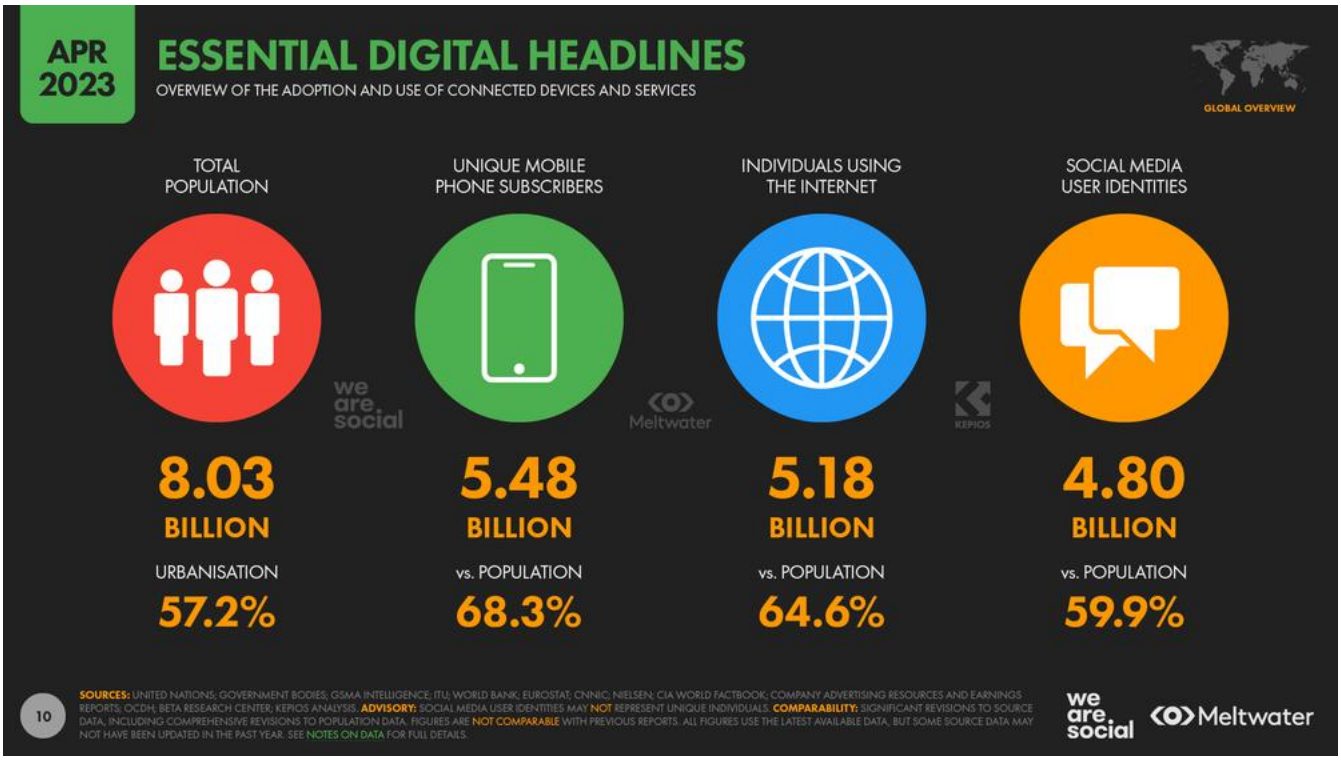


Рисунок 1.5 – Кількість користувачів та розподіл трафіку мережі Інтернет

Причина #3 Мобільний трафік переходить до інтернет-месенджерів.

Довести те, що все більше людей з роками стало користуватися саме інтернет мессенджерами просто. Це можна зробити на прикладі таких компаній як Facebook та WhatsApp. Facebook придбала популярний месенджер та допомогла йому стати ще популярним. За 2013 рік збитки компаній операторів стільникового зв’язку сягнули 32.5 млн доларів. Це сталося через те, що користувачі стали частіше обирати саме інтернет-спілкування замість засобів стільникового зв’язку.

Через збільшення доступності пристроїв для зв’язку через інтернет та збільшення мобільного інтернет-трафіку у світі стає зрозумілим, що нині онлайн-спілкування – найпоширеніший спосіб швидкого зв’язку між людьми на планеті.

1.2 Огляд найпоширеніших застосунків та онлайн-систем у сфері спілкування

За результатами дослідження трьома найпопулярнішими мобільними мессенджерами у світі за минулий рік є:

1) WhatsApp: займає першу позицію у 63 країнах, серед них: Аргентину, Австрію, Бахрейн, Бельгію, Болівію, Боснію і Герцеговину, Бразилію, Чилі, Коста-Рику, Хорватію, Чеську Республіку, Колумбію, Домініканську Республіку, Еквадор, Єгипет, Сальвадор, Фінляндію, Францію, Німеччину, Гану, Гонконг, Індію, Індонезію, Ірландію, Ізраїль, Італію, Ямайку, Кувейт, Кенію, Ліван, Македонію, Малайзію, Мексику, Марокко, Непал, Нідерланди, Нігерію, Оман, Пакистан, Панаму, Парагвай, Перу, Португалію, Пуерто-Рико, Катар, Румунію, Росію, Руанду, Саудівську Аравію, Сінгапур, Словенію, ПАР, Іспанію, Шрі-Ланку, Швейцарію, Танзанію, ОАЕ, Великобританію, Уругвай, Венесуелу та Ємен.
2) Facebook Messenger: зберігає свою позицію з показником у 16 країн, серед яких: Алжир, Австралія, Бангладеш, Канада, Данія, Угорщина, Литва, Нова Зеландія, Філіппіни, Польща, Словаччина, Швеція, Тайвань, Таїланд, Туніс і США.
3) Telegram: є найпопулярнішим мобільним застосунком для спілкування у 10 країнах: Вірменія, Камбоджа, Ірак, Казахстан, Киргизія, Іорданія, Латвія, Молдова, Україна і Узбекистан[5].

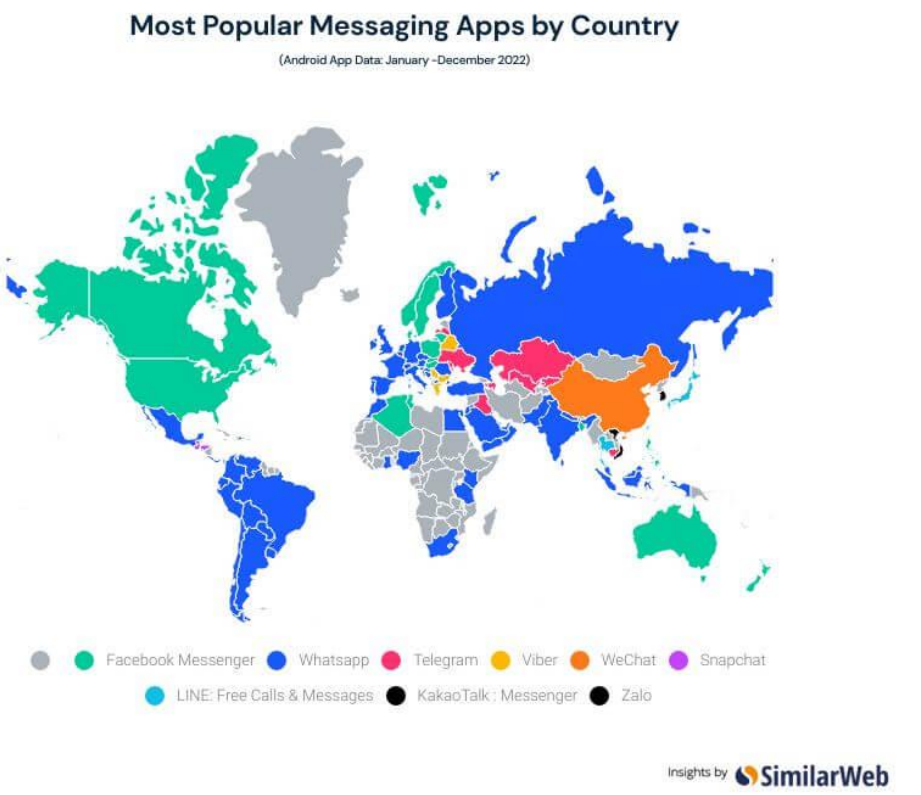


Рисунок 1.6 – Мапа популярності месенджерів у країнах світу

Таблиця 1.2 – Порівняльна характеристика активності месенджерів

	Facebook	WhatsApp	Telegram
MAU (Millions)	763	1349	412.6
DAU (Millions)	404.6	1221	196
% of users active daily	53.03	90.51	47.50
Sessions per user	9.94	22.97	8.7
Average session time (Mins)	1.54	2.35	2.24
Total session time (Mins)	18.53	59.32	20.56

WhatsApp займає однозначне лідерство з кількості MAU (monthly active users) - активних користувачів за місяць, кількості активних користувачів за день та проценту активних користувачів за день та кількості сесій одного користувача за день. Усе це призводить до того, що середній користувач WhatsApp проводить майже годину в день у застосунку.

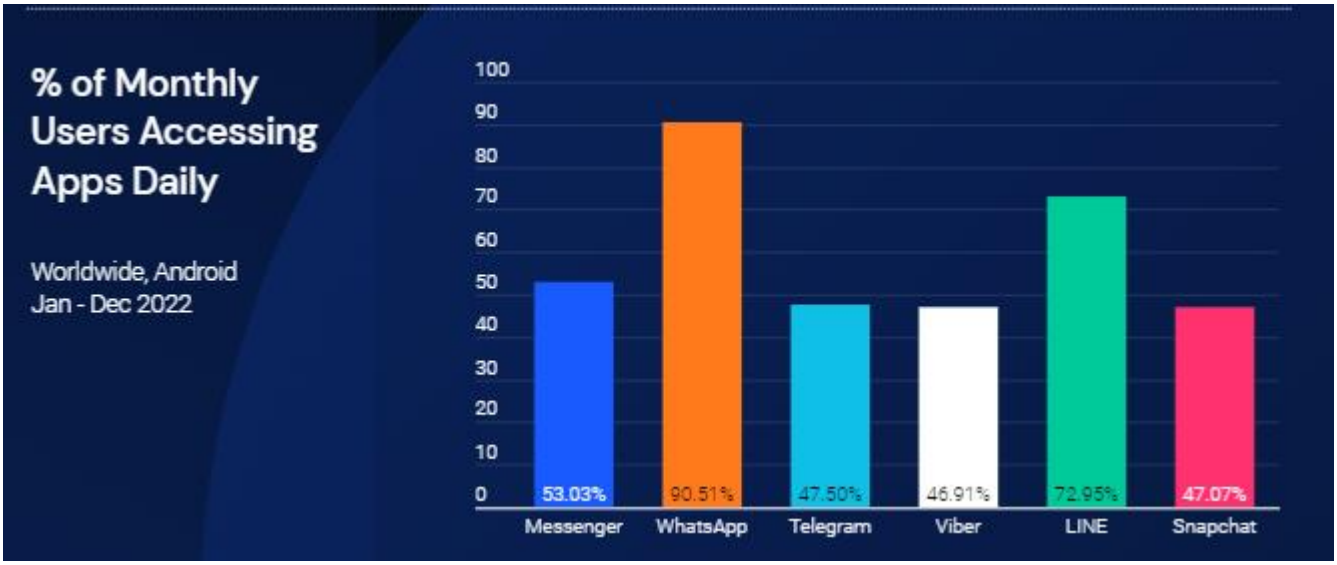


Рисунок 1.7 – Відсоток користувачів за місяць, які заходять у застосунок кожен день

За даними за 2022 рік можна зробити висновок що половина чи більше зареєстрованих користувачів у мессенджерах користуються ним кожен день.

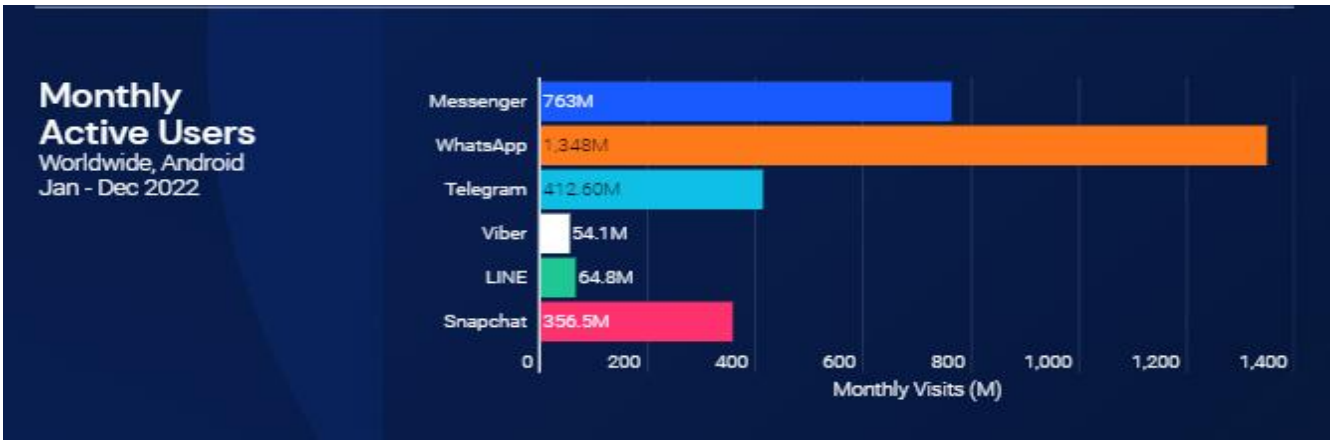


Рисунок 1.8 – Кількість користувачів за місяць у популярних мессенджерах.

Як можна побачити з діаграми – WhatsApp займає лідерство з 1.3 мільярда активних користувачів щомісячно, це 16,25% від популяції планети тільки в одному месенджері.

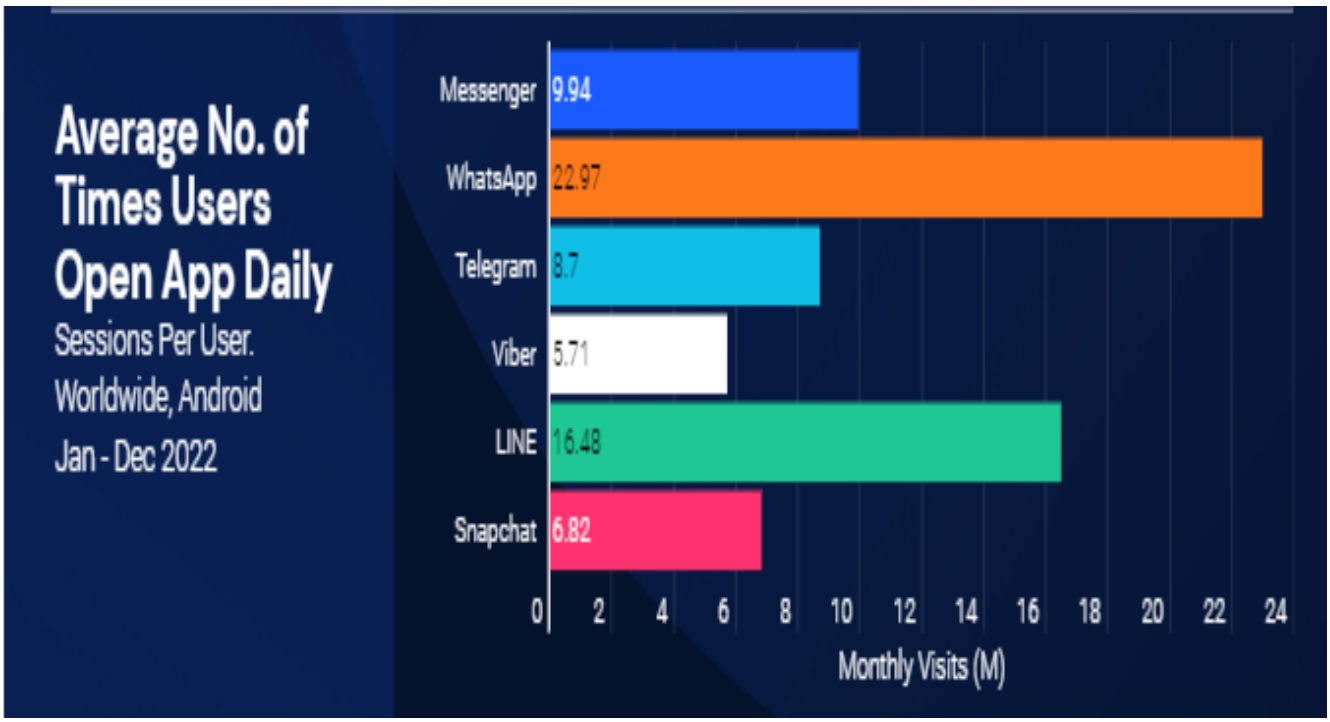


Рисунок 1.9 – Середня кількість сесій користування застосунком за день.

З цих даних можна зробити висновок, що користувачі мобільних застосунків – месенджерів активно користуються чатами протягом дня для того щоб підтримувати зв’язок у різних сферах сучасного життя.

У нашій країні на даний момент найпопулярнішим мобільним застосунком для спілкування є Telegram, тому проаналізуємо його як один з прикладів успішного програмного рішення у сфері спілкування.

Telegram - це популярний мобільний чат-застосунок, який надає можливості миттєвого обміну повідомленнями, відео-викликами, спільною роботою над файлами та багато іншого. Особливості Telegram полягають у його безпекових заходах, широких можливостях налаштувань та екосистемі розширень.

Рисунок 1.10 – Логотип Telegram

Окрім обміну повідомленнями у діалогах та групах, у месенджері можливо зберігати будь-яку кількість файлів, вести мікроблоги та створювати і використовувати ботів.

Телеграм-бот - це автоматизована програма, яка працює в месенджері Telegram і взаємодіє з користувачами через текстові повідомлення. Він може виконувати різні завдання та функції, починаючи від відповіді на запитання і надання інформації до виконання складніших дій.

Основні особливості телеграм-ботів:

- автоматизація: Телеграм-боти виконують завдання автоматично, без необхідності присутності людини. Вони можуть відповідати на питання, обробляти запити та виконувати інші дії, підтримуючи взаємодію з користувачами у режимі реального часу;
- різноманітні функції: Телеграм-боти можуть мати різноманітний функціонал. Вони можуть надавати новини, прогноз погоди, розклад занять,

генерувати випадкові фрази, виконувати конвертацію валют, надавати ігрові елементи та багато іншого. Обмеження функціональності залежить від розробника бота;

- взаємодія з користувачами: Користувачі можуть спілкуватися з телеграм-ботами, відправляючи їм текстові повідомлення. Бот може розпізнавати команди, запитання або інструкції від користувачів і реагувати на них відповідним чином;
- розвиток та інтеграція: Телеграм-боти можуть бути розширені та розвиватися шляхом додавання нових функцій, інтеграції зі зовнішніми сервісами та розширенням можливостей взаємодії з користувачами.

Телеграм-боти стали популярними завдяки своїй простоті, широкому спектру функцій та можливості автоматизації різних задач. Вони знаходять застосування в різних галузях, включаючи бізнес, освіту, медіа, розваги та інші.

Цей застосунок був запущений у 2013 році Павлом Дуровим. Спочатку Telegram був розроблений як альтернатива іншим месенджерам, з фокусом на приватність та безпеку даних. У квітні 2014 року Telegram оголосив про запуск власного API, що дозволило розробникам створювати сторонні застосунки та ботів для платформи.

- API (Application Programming Interface) - це набір визначень та протоколів, які дозволяють різним програмним застосунком взаємодіяти між собою. API визначає, які функції та можливості можуть бути використані програмними застосунків, як обмінюватись даними та які правила використовувати при такій взаємодії.

Основні особливості API:

- спрощена взаємодія: API надає спрощений спосіб взаємодії між програмами. Він визначає, які команди або запити можуть бути виконані, які дані необхідні для передачі та які результати можуть бути отримані;
- стандартизація: API дозволяє різним розробникам створювати програми, використовуючи спільні стандарти і протоколи. Це полегшує розробку та інтеграцію програмних рішень з іншими застосунків;
- розширення функціоналу: За допомогою API розробники можуть розширити функціональні можливості своїх програм, використовуючи функції та сервіси, надані іншими програмами або платформами;
- взаємодія з веб-сервісами: Багато API дозволяють взаємодіяти з веб-сервісами, такими як соціальні мережі, платіжні системи, картографічні сервіси та багато інших. Це дозволяє програмам отримувати доступ до зовнішніх ресурсів та послуг.

API може бути представлено у вигляді набору документації, бібліотеки програмного забезпечення, веб-сервісу або набору протоколів. Він забезпечує інтерфейс для комунікації між різними програмними компонентами, дозволяючи їм обмінюватись даними та функціональністю.

Telegram використовую клієнт – серверну архітектуру, де клієнти здійснюють з’єднання з серверами Telegram для обміну даними. Застосунок використовує розподільну мережу серверів, що забезпечує швидкий доступ та стабільну роботу застосунку.

Клієнт-серверна архітектура - це модель розподіленого обчислення, в якій взаємодіють два основних компонента: клієнт і сервер. Клієнт - це програма або

пристрій, який надає користувачеві інтерфейс для взаємодії з системою. Сервер - це програма або комп'ютер, який надає ресурси та послуги клієнтам.

Основні принципи клієнт-серверної архітектури:

- розділення обов'язків: Клієнт відповідає за взаємодію з користувачем та обробку локальних даних. Сервер відповідає за обробку запитів клієнта, зберігання та обробку даних, надання послуг та відповідей на запити;
- комунікація: Клієнт і сервер обмінюються даними та командами за допомогою мережевих протоколів, таких як HTTP, TCP/IP або інші. Клієнт надсилає запити серверу, а сервер обробляє ці запити та надсилає відповіді клієнту;
- масштабованість: Клієнт-серверна архітектура дозволяє гнучко масштабувати систему. Сервер може обробляти запити великої кількості клієнтів шляхом розподілу навантаження, використання кластеризації або інших методів масштабування;
- безпека: Клієнт-серверна архітектура дозволяє контролювати доступ до ресурсів та даних. Сервер може встановлювати правила аутентифікації та авторизації для клієнтів, щоб забезпечити безпеку та конфіденційність інформації.

Клієнт-серверна архітектура використовується в багатьох сферах, включаючи веб-застосунки, мобільні застосунки, бази даних, хмарні сервіси та багато інших систем. Вона дозволяє ефективно організовувати взаємодію між різними компонентами системи та забезпечує розширення та розподілення функціональності.

Telegram використовує власний протокол обміну даними MTProto, який забезпечує шифрування трафіку та безпеку передачі повідомлень[6].

Протокол MTProto є протоколом передачі даних, розробленим компанією Telegram для використання в своєму месенджері. Основна мета цього протоколу - забезпечити шифровану та безпечну комунікацію між користувачами.

Основні характеристики протоколу MTProto:

- шифрування: Протокол MTProto використовує симетричне шифрування для захисту конфіденційності даних. Дані, включаючи повідомлення, медіафайли та метадані, шифруються за допомогою AES (Advanced Encryption Standard) з ключем, який обмінюється між користувачами під час процесу аутентифікації;
- стійкість до блокування: MTProto розроблений з метою уникнути блокування інтернет-провайдерами та цензурою. Він використовує динамічну зміну портів та протоколів для унеможливлення виявлення та блокування трафіку месенджера;
- ефективність: MTProto оптимізований для передачі даних на мобільних пристроях з обмеженими ресурсами. Він використовує компресію даних та інші оптимізації для забезпечення швидкої та ефективної передачі повідомлень.

Протокол призначений для доступу до серверного API з застосунків, що працюють на мобільних пристроях (слід зауважити, що веб-браузер не є таким застосунком). Протокол поділяється на три майже незалежні компоненти:

- високорівневий компонент (мова запитів API): визначає метод перетворення запитів і відповідей API в бінарні повідомлення;
- криптографічний (авторизаційний) шар: визначає метод шифрування повідомлень перед їх передачею через транспортний протокол;

- транспортний компонент: визначає метод передачі повідомлень між клієнтом і сервером через існуючий мережевий протокол (наприклад, HTTP, HTTPS, WS (прості WebSockets), WSS (WebSockets через HTTPS), TCP, UDP).

Історія виникнення протоколу MTProto пов'язана з розробкою Telegram. Протокол був розроблений Павлом Дуровим та його командою для забезпечення безпеки та приватності користувачів. Протокол поступово розвивався та вдосконалювався з метою поліпшення шифрування та ефективності комунікації.

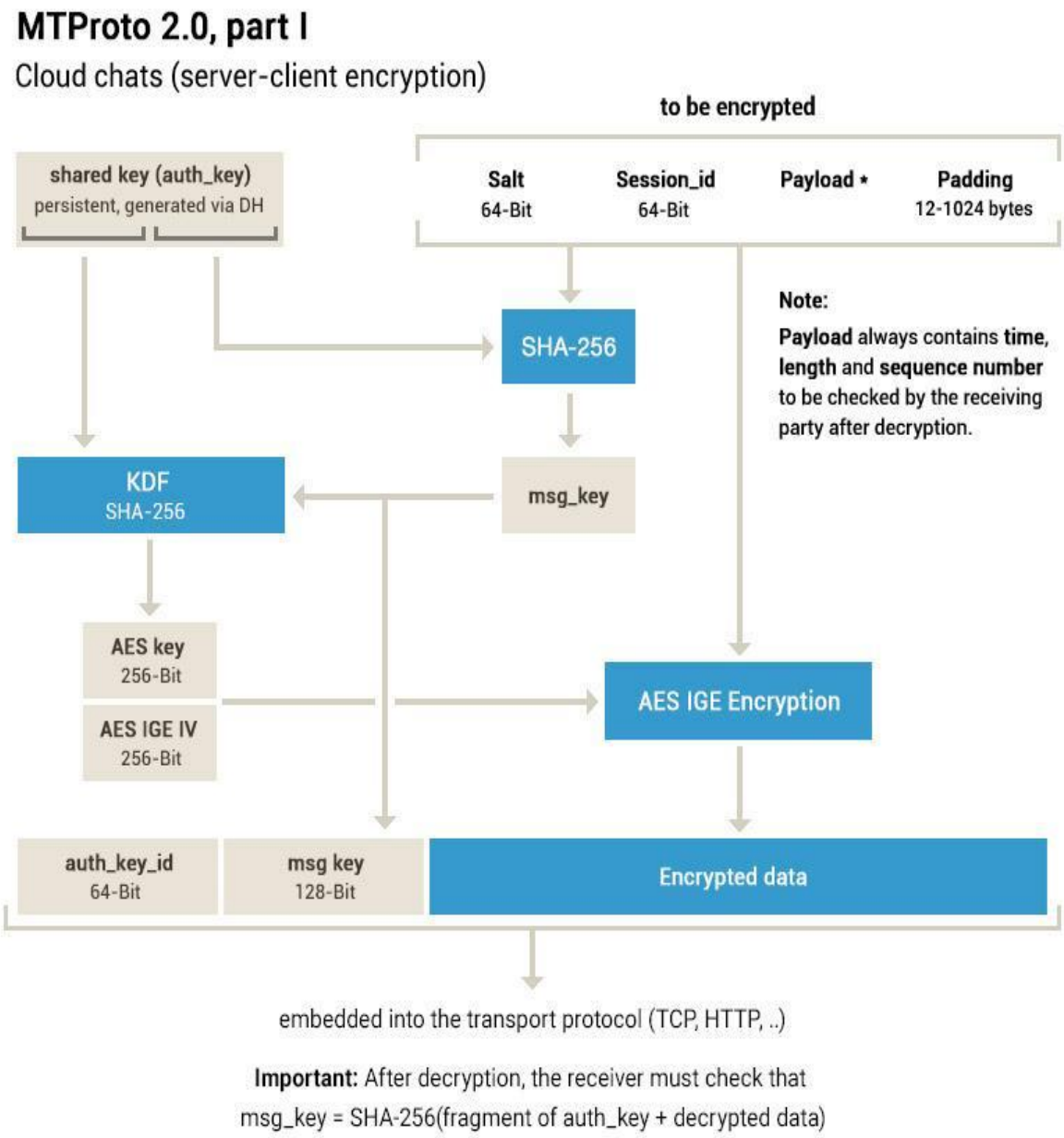


Рисунок 1.11 – Схема роботи протоколу MTProto

1.3 Підходи до розробки застосунку - мессенджера

1.3.1 Підходи до розробки на основі протоколу

Розробка мобільного застосунку для спілкування – комплексне завдання, яке залежить від багатьох факторів – потреб проекту, ресурсів та вимог до швидкості розробки та підтримки. Вирізняються декілька варіантів архітектур таких застосунків та підходів до розробки, наприклад: розробка на основі власного

протоколу, використання відкритих протоколів, використання платформи мессенджера(іншого) та використання специфічних фреймворків та SDK.

Протокол - це набір правил, процедур та форматів, які визначають спосіб комунікації та взаємодії між двома або більше системами. В контексті програмного забезпечення, протокол визначає правила обміну даними між програмами, сервісами або пристроями. Протоколи встановлюють правила, які потрібно дотримуватися під час обміну даними, включаючи структуру повідомлень, формат передачі, кодування даних, способи аутентифікації та шифрування, а також обробку помилок і контрольні суми. Протоколи можуть бути використані в різних галузях, включаючи мережеві комунікації, веб-протоколи, мобільні застосунки, месенджери, соціальні мережі та багато іншого. Наприклад, HTTP (Hypertext Transfer Protocol) є протоколом, який використовується для передачі веб-сторінок, SMTP (Simple Mail Transfer Protocol) - для відправки електронних листів, а TCP/IP - для мережевої комунікації[7]. Протоколи дозволяють різним системам спілкуватися та взаємодіяти, забезпечуючи стандартизований та надійний обмін даними. Вони грають важливу роль у розробці програмного забезпечення, де різні компоненти повинні спілкуватися один з одним, дотримуючись встановлених правил протоколу.

Розглянемо кожен з підходів до розробки застосунка.

1) Розробка мобільного мессенджера на основі власного протоколу комунікації є одним з підходів до створення таких застосунків. Цей підхід надає повний контроль над функціональністю, безпекою та розширюваністю системи.
   Основні кроки розробки:

   - аналіз вимог: Визначення функціональних та нефункціональних вимог до мессенджера, таких як обмін повідомленнями, групові чати, файловий обмін, захист даних тощо;

- проектування протоколу: Розробка власного протоколу комунікації, який визначає правила і формати обміну даними між клієнтом і сервером. Це включає визначення структури повідомлень, механізмів аутентифікації та шифрування даних;
- розробка клієнтської програми: Створення мобільного застосунку-клієнта для мобільних пристроїв, який взаємодіє з сервером за допомогою власного протоколу. Це включає реалізацію інтерфейсу користувача, обробку повідомлень, керування контактами та інші функції;
- розробка серверної частини: Створення серверного застосунку, який обробляє запити від клієнтів, зберігає дані, керує користувачами та забезпечує комунікацію між клієнтами. Це може включати в себе використання бази даних для зберігання повідомлень та інформації про користувачів;
- тестування та підтримка: Виконання ряду тестів для перевірки функціональності, безпеки та швидкодії мобільного мессенджера. Після випуску продукту в експлуатацію необхідно забезпечити його підтримку, виправлення помилок та розширення функціональності за потреби.

Переваги розробки на основі власного протоколу включають повний контроль над системою, можливість реалізувати унікальні функції та забезпечити високий рівень безпеки. Однак, цей підхід також вимагає значних зусиль у розробці, тестуванні та підтримці, а також може потребувати експертизи у протоколах комунікації та криптографії[8].

2) Розробка на основі відкритого протоколу є підходом, при якому мобільний мессенджер реалізує свою функціональність та комунікацію засобами вже існуючого відкритого протоколу.

Відкритий протокол (Open Protocol) - це протокол, який опублікований і доступний для використання та імплементації всім бажаючим без обмежень та ліцензійних обов'язків. Відкриті протоколи зазвичай розробляються та підтримуються спільнотою розробників, організаціями або стандартизаційними органами.

Основні риси відкритих протоколів:

- відкритість: Відкриті протоколи мають відкритий та доступний для всіх документацію, специфікації та референсні реалізації. Це дає можливість розробникам створювати власні імплементації протоколу та взаємодіяти з іншими системами;
- стандартизація: Відкриті протоколи часто підлягають стандартизації організаціями, такими як IETF (Internet Engineering Task Force) або W3C (World Wide Web Consortium). Це допомагає забезпечити їхню сумісність та використання в широкому спектрі систем;
- інтероперабельність: Відкриті протоколи сприяють інтероперабельності між різними системами та платформами. Це дозволяє різним розробникам та організаціям взаємодіяти та обмінюватися даними без обмежень;
- розвиток спільноти: Відкриті протоколи залучають спільноту розробників, яка активно працює над їх покращенням, додатковими функціями та виправленнями помилок. Це сприяє швидкому розвитку протоколу та вирішенню проблем.

Приклади відкритих протоколів включають в себе: HTTP (Hypertext Transfer Protocol), SMTP (Simple Mail Transfer Protocol), JSON-RPC (Remote Procedure Call) та інші. Відкриті протоколи відіграють важливу роль у сприянні вільному обміну даними між різними системами та платформами.

Основними перевагами розробки застосунку на основі відкритого протоколу є: стандартизація – використання відкритого протоколу змушує розробляти проект відповідно до визначеного стандарту, що спрощує взаємодію з іншими системами; заощадження часу, тому-що не потрібно розробляти власний протокол з нуля; гнучкість – відкриті протоколи надають широкий спектр функцій та можливостей, що дозволяє легко адаптувати месенджер під потреби проекту; екосистема – використання відкритого протоколу зазвичай відкриває доступ до готової екосистеми з документацією, іншими розробниками та готовими інструментами для взаємодії[9].

3) Використання платформи мессенджера означає створення мобільного застосунка або системи комунікації на основі вже існуючого мессенджера. Замість розробки повноцінного мобільного застосунка з нуля, розробники можуть використовувати готову платформу мессенджера, яка надає необхідні функції та інструменти для обміну повідомленнями, спілкування та передачі даних.

   Переваги використання платформи мессенджера:

   - швидкий старт: Використання платформи мессенджера дозволяє розробникам швидко створити функціональний мобільний застосунок, не витрачаючи час на розробку базових функцій комунікації;
   - готовий користувацький шар: Платформи месенджерів часто мають вже готовий користувацький інтерфейс, який можна налаштувати та пристосувати до потреб проекту. Це забезпечує зручне та знайоме середовище для користувачів;
   - розширені можливості: Платформи месенджерів можуть мати вбудовані функції для обміну файлами, відео та голосовими повідомленнями, груповими чатами та іншими функціями, що розширюють можливості застосунка;

- готові інтеграції: Платформи месенджерів часто надають API або SDK для зручної інтеграції з іншими сервісами та застосунків. Це дозволяє розробникам використовувати додаткові функції та сервіси, щоб розширити можливості свого застосунка;
- масштабованість: Використання платформи мессенджера дозволяє розробникам впроваджувати свій застосунок на вже існуючій популярній платформі з великою кількістю активних користувачів, що сприяє швидкому зростанню та масштабуванню користувацької бази.

Проте, варто враховувати, що використання такої платформи може мати обмеження щодо налаштування та індивідуальності, а також може залежати від умов використання та ліцензійних умов платформи.

4) Використання фреймворків та SDK (Software Development Kit) є популярним підходом при розробці мобільних месенджерів. Фреймворки та SDK надають набір інструментів, бібліотек та готових рішень, що допомагають спростити розробку застосунків та прискорити процес розробки. Основні переваги використання фреймворків та SDK у розробці мобільних месенджерів включають:

- готові компоненти: Фреймворки та SDK надають готові компоненти, такі як елементи інтерфейсу, модулі для роботи з повідомленнями, медіа, шифруванням тощо. Це дозволяє розробникам ефективно використовувати готові рішення та скоротити час розробки;
- кросс-платформеність: Деякі фреймворки, наприклад, React Native або Flutter, дозволяють розробляти мобільні застосунки, які працюють на різних платформах, таких як iOS та Android[10]. Це дозволяє зекономити час та зусилля, оскільки розробка ведеться одночасно для кількох платформ;

- швидкість розробки: Використання фреймворків та SDK дозволяє прискорити процес розробки, оскільки розробникам не потрібно писати весь код з нуля. Готові компоненти та інструменти допомагають розробникам швидко створювати функціональність мессенджера;
- підтримка та оновлення: Фреймворки та SDK зазвичай мають активну спільноту розробників, яка надає підтримку, документацію та оновлення. Це дозволяє розробникам легко знайти рішення для своїх проблем та бути в курсі останніх трендів та вдосконалень.

Приклади фреймворків та SDK, що використовуються у розробці мобільних месенджерів, включають Firebase (для роботи з базою даних та повідомленнями), Twilio (для відео- та голосових дзвінків), SendBird (для реального часу спілкування) та багато інших.

1.3.2 Підходи на основі архітектури

При розробці мобільних месенджерів, існує кілька архітектурних підходів, які можуть бути використані для структуризації та організації застосунку. Ось декілька типових архітектурних підходів для мобільних месенджерів:

1. Клієнт-серверна архітектура: Цей підхід передбачає наявність централізованого сервера, який відповідає за зберігання повідомлень, передачу даних між користувачами та обробку різних комунікаційних операцій. Клієнтська частина застосунку (мобільний застосунок) взаємодіє з сервером для отримання та надсилання повідомлень[11].

   Основні принципи клієнт-серверної архітектури:

- розділення обов'язків: клієнт відповідає за взаємодію з користувачем та обробку локальних даних. Сервер відповідає за обробку запитів клієнта, зберігання та обробку даних, надання послуг та відповідей на запити;
- комунікація: клієнт і сервер обмінюються даними та командами за допомогою мережевих протоколів, таких як HTTP, TCP/IP або інші. Клієнт надсилає запити серверу, а сервер обробляє ці запити та надсилає відповіді клієнту;
- масштабованість: клієнт-серверна архітектура дозволяє гнучко масштабувати систему. Сервер може обробляти запити великої кількості клієнтів шляхом розподілу навантаження, використання кластеризації або інших методів масштабування;
- безпека: клієнт-серверна архітектура дозволяє контролювати доступ до ресурсів та даних. Сервер може встановлювати правила аутентифікації та авторизації для клієнтів, щоб забезпечити безпеку та конфіденційність інформації.

Клієнт-серверна архітектура використовується в багатьох сферах, включаючи веб-застосунки, мобільні застосунки, бази даних, хмарні сервіси та багато інших систем. Вона дозволяє ефективно організовувати взаємодію між різними компонентами системи та забезпечує розширення та розподілення функціональності.

2. P2P (Peer-to-Peer) архітектура: У цьому підході, мобільні пристрої можуть взаємодіяти один з одним без проміжного сервера. Вони можуть встановлювати пряме з'єднання між собою, обмінюватися повідомленнями та даними. Це може забезпечити швидку та безпечну

передачу повідомлень без залучення третіх сторін[12]. Основні характеристики P2P архітектури включають:

- безпосередня комунікація: у P2P архітектурі мобільних месенджерів пристрої можуть спілкуватися безпосередньо між собою, обмінюючись повідомленнями, медіафайлами та іншими даними. Це дозволяє забезпечити швидку та пряму комунікацію між користувачами без проміжних серверів;
- децентралізованість: кожен пристрій у P2P мережі мобільного мессенджера виконує роль як сервера, так і клієнта, що призводить до децентралізованої структури. Це означає, що немає центрального сервера, який контролює всі комунікації, а замість цього комунікація здійснюється безпосередньо між пристроями;
- резервування ресурсів: у P2P архітектурі ресурси (такі як пропускна здатність мережі, потужність обчислювальних пристроїв) резервуються між усіма пристроями в мережі. Кожен пристрій вносить свій внесок у комунікацію та обмін даними;
- склеювання та гнучкість: P2P архітектура дозволяє гнучко масштабувати мережу. При додаванні нових пристроїв до мережі збільшується її потужність та можливості для комунікації.

Підходи до реалізації P2P архітектури включають використання технологій, таких як WebRTC (Web Real-Time Communication), Bluetooth, Wi-Fi Direct та інші, що дозволяють безпосередню комунікацію між пристроями у мережі.

3. Hybrid архітектура: Цей підхід поєднує клієнт-серверну та P2P архітектури. Деякі комунікаційні операції можуть відбуватися безпосередньо між мобільними пристроями (P2P), тоді як інші можуть

потребувати взаємодії з сервером (клієнт-сервер). Цей підхід базується на використанні фреймворків та інструментів, які дозволяють розробляти мобільні застосунки, використовуючи веб-технології, такі як HTML, CSS та JavaScript, а також можливості нативних платформ[13]. Основними характеристиками Hybrid-архітектури є:

- нативний контейнер: застосунок розробляється з використанням фреймворків, які надають нативний контейнер, який збудував веб-код у мобільну платформу. Цей контейнер дозволяє використовувати нативні можливості пристрою, такі як камера, геолокація та сповіщення;
- веб-переглядач: використовуються вбудовані веб-переглядачі для відображення веб-інтерфейсу застосунка. Це дозволяє використовувати веб-технології для створення інтерфейсу користувача та взаємодії з ним;
- гібридний функціонал: завдяки гібридній архітектурі, мобільний мессенджер може мати як веб-засновані елементи, так і нативні функціональні можливості. Це дозволяє поєднувати переваги швидкості та доступу до пристрою з веб-технологіями для швидкого розвитку та платформо незалежності;
- оновлення через Інтернет: За допомогою гібридної архітектури можна здійснювати оновлення застосунків через Інтернет без необхідності завантажувати нові версії з магазинів застосунків.

4. Архітектура "Backend-as-a-Service" (BaaS) в розробці мобільних месенджерів передбачає використання стороннього сервісу, який надає готовий backend (серверну частину) для застосунків. У цьому підході

розробник зосереджується на створенні клієнтської частини застосунка, в той час як всі серверні операції, такі як зберігання даних, автентифікація користувачів та обробка повідомлень, виконуються за допомогою зовнішнього сервісу. Основними особливостями архітектури BaaS є:

- серверні функції: BaaS-платформи надають набір готових серверних функцій, які можуть бути використані для реалізації різноманітного функціоналу, такого як обробка повідомлень, керування користувачами та операції з базою даних;
- управління даними: BaaS-платформи забезпечують можливість зберігання та керування даними застосунка. Це може включати роботу з базами даних, хмарним сховищем та іншими засобами зберігання;
- автентифікація та безпека: BaaS-платформи надають механізми аутентифікації користувачів та захисту даних, включаючи різні методи аутентифікації, шифрування та контроль доступу;
- інтеграція з іншими сервісами: BaaS-платформи часто надають можливість інтеграції з іншими сторонніми сервісами, такими як соціальні мережі, платіжні системи тощо.

Використання архітектури BaaS дозволяє розробникам швидко створювати мобільні мессенджери, зменшуючи необхідність у власному серверному інфраструктурі та спрощуючи розробку та підтримку застосунків[14].

Ці архітектурні підходи можуть бути комбіновані або адаптовані залежно від конкретних вимог та функціональності мобільного месенджера. Кожен підхід має свої переваги та особливості, і вибір залежить від конкретних потреб проекту та обмежень.

1.4. Формування завдання на розробку

Завданням бакалаврської роботи є побудова та розробка мобільного чат-застосунку з використанням технології Xamarin. Врахувавши описані вище особливості підходів та архітектури, для розробки власного застосунку вважаю за потрібне обрати архітектуру BaaS з використанням Google Firebase. Для подальшого виконання поставленої мети вважаю за потрібне:

- Визначення вимог: Встановлення функціональних та нефункціональних вимог до чат-застосунку, таких як можливість обміну повідомленнями, реєстрація користувачів, реалізація групових чатів тощо.
- Проектування архітектури: Розроблення архітектури чат-застосунку з використанням технології Xamarin, включаючи розподілення клієнтської та серверної частин, вибір протоколів комунікації та дизайн інтерфейсу користувача.
- Розробка клієнтської частини: Реалізація клієнтської частини чат-застосунку з використанням Xamarin, включаючи створення користувацького інтерфейсу, реалізацію функцій обміну повідомленнями, керування контактами та інші необхідні функції.
- Розробка серверної частини: Реалізація серверної частини чат-застосунку, яка забезпечує обробку та збереження повідомлень, аутентифікацію користувачів та керування груповими чатами.
- Тестування та налагодження: Виконання тестування чат-застосунку з метою виявлення та усунення помилок, а також налагодження і вдосконалення функціональності

Висновки до розділу

Був описаний успішний проект у сфері спілкування за допомогою мобільних пристроїв. Були наведені дані, які демонструють щомісячне збільшення користувачів інтернету, міграцію трафіку з наземних та стільникових систем до мережі інтернет. Були наведені та порівнянні дані про збільшення доступності

мобільних пристроїв, зменшення їх вартості на фоні суттєвого збільшення середньої потужності. Була наведена статистика поведінки користувачів у мобільних мессенджерах, були продемонстровані тенденції до збільшення аудиторії та збільшення часу, який користувачі проводять у таких застосунках. На основі цього, вважаю розробку мобільного застосунку-чату актуальною.

Були наведені підходи до розробки архітектури таких застосунків, проаналізовані переваги кожного з них та обрано один з них для подальшої розробки завдання бакалаврської роботи.

Визначені задачі та вимоги для подальшого процесу реалізації застосунку.

РОЗДІЛ 2

АНАЛІЗ ПРОГРАМНО-ТЕХНОЛОГІЧНИХ РІШЕНЬ ДЛЯ РОЗРОБКИ МОБІЛЬНОГО ЗАСТОСУНКУ ЧАТУ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ XAMARIN

2.1 Опис мов програмування та засобів для розробки мобільного застосунку за допомогою технології Xamarin

Для успішного виконання мого завдання до бакалаврської роботи я повинен розробити програмне рішення у сфері спілкування для мобільних пристроїв з використанням технології Xamarin.

Xamarin – це інтегроване середовище розробки (IDE) та платформа для створення кросплатформенних мобільних застосунків. Вона базується на мові програмування C# і використовує спільний код для розробки застосунків для різних мобільних платформ, таких як iOS, Android та Windows. Для того щоб зрозуміти можливості фреймворку, потрібно спочатку проаналізувати можливості мови, на якій з його допомогою розробляються мобільні застосунки. Застосунки Xamarin розробляються мовою C#[15].

C# ("Сі-шарп") - це сучасна, об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. Вона була вперше представлена у 2000 році і стала однією з основних мов для розробки програмного забезпечення на платформі .NET.

Історія розробки C# почалася в 1999 році, коли Microsoft утворила команду розробників, яка працювала над створенням нової мови програмування. Основними розробниками були Андерс Гейлсберг та їхній колектив. Метою було створення мови, яка поєднує простоту та продуктивність мови C++ з легкістю використання мови Java. C# отримав вплив від різних мов програмування, включаючи C++, Java, Delphi та інші. Він поєднує в собі синтаксичні особливості мови C++ і засади програмування мови Java.

Мова C# пропонує багато можливостей для розробки програмного забезпечення, включаючи:

- об'єктно-орієнтований підхід: C# підтримує об'єктно-орієнтоване програмування, що дозволяє створювати класи, об'єкти, успадкування, поліморфізм та інші концепції ООП;
- можливості мови програмування: C# має багатий набір функцій, таких як типи даних, управління пам'яттю, засоби обробки винятків, делегати, події, лямбда-вирази, LINQ (Language-Integrated Query) та багато іншого;
- багатопотокова підтримка: C# надає можливості для роботи з потоками, що дозволяє ефективно використовувати багатоядерні процесори та створювати багатопотокові програми;
- широке застосування: C# використовується для розробки різноманітних програм, включаючи десктопні застосунки, веб-програми, мобільні застосунки та інші;
- інтеграція з .NET: C# є однією з мов, які підтримуються платформою .NET. Це означає, що ми можемо використовувати бібліотеки класів .NET, що забезпечують різноманітні функціональні можливості, такі як робота з базами даних, мережеве програмування, робота з файлами, шифрування даних тощо[16].

C# став популярною мовою програмування завдяки своїм можливостям, великій підтримці від Microsoft, активній спільноті розробників та широкому застосуванню в різних сферах програмування. Він продовжує розвиватися та оновлюватися з кожним новим випуском, надаючи розробникам зручний та потужний інструмент для створення програмного забезпечення.

Актуальною версією мови C# є C# 11, яка вийшла 8 листопада 2022 року разом з релізом .NET 7. Остання версія мови представила широкий спектр оновлень, додавши нові елементи та атрибути, такі як:

- raw string literals (Необроблені рядкові літерали);
- universal math support (Підтримка універсальної математики);
- universal attributes (Універсальні атрибути);
- UTF-8 string literals (Рядкові літерали UTF-8);
- new lines in string interpolation expressions (Нові рядки у виразах інтерполяції рядків);
- list patterns (Шаблони списків);
- local file types (Локальні типи файлів);
- required members (Обов'язкові члени);
- default-initialized structures (Автоматично ініціалізовані структури за замовчуванням);
- matching Span<char> patterns to a string constant (Співставлення шаблонів Span<char> з константою string);
- expanded nameof scope (Розширена область для nameof);
- numeric intPtr (Числовий intPtr);
- ref fields and scoped ref (ref поля та області з ref);
- improved method group conversion for delegates (Покращене перетворення груп методів для делегатів);
- warning signal 7 (Попереджувальний сигнал 7)[17].

Мова програмування C# була спеціально розроблена групою інженерів у 1999-2001 роках під керівництвом Андерса Хейлсберга та Скотта Вільтаумота як «мова розробки застосунків для платформи Microsoft, .NET та .NET Core. Для успішної реалізації завдання бакалаврської роботи я повинен ознайомитися із функціоналом платформи .NET.

.NET є платформою, розроблено компанією Microsoft. Вона надає середовище для створення, розгортання та виконання різноманітних програм, включаючи десктопні застосунки, веб-застосунки, мобільні застосунки та хмарні сервіси. Можна виділити наступні її наступні основні риси:

- підтримка кількох мов - основою платформи є загальномовне середовище виконання Common Language Runtime (CLR), завдяки чому .NET підтримує кілька мов: крім C#, це також VB.NET, C++, F#, а також різні діалекти інших мов, що підтримуються .NET, наприклад, Delphi.NET. При компіляції коду на будь-якій з цих мов він перетворюється в збірку на загальній мові CIL (Common Intermediate Language) - своєрідний асемблер платформи .NET. Тому за певних умов ми можемо створювати окремі модулі одного застосунка на різних мовах;
- кросплатформність - .NET є переносною платформою (з деякими обмеженнями). Наприклад, остання версія платформи на сьогоднішній день - .NET 7 - підтримується на більшості сучасних операційних систем Windows, MacOS, Linux. З використанням різних технологій на платформі .NET можна розробляти застосунки на мові C# для різних платформ - Windows, MacOS, Linux, Android, iOS, Tizen;
- потужна бібліотека класів - .NET надає єдину для всіх підтримуваних мов бібліотеку класів. І незалежно від того, який застосунок ми збираємося

писати на C# - текстовий редактор, чат або складний веб-сайт - ми обов'язково використаємо бібліотеку класів .NET;

- різноманітні технології - загальномовне середовище виконання CLR та базова бібліотека класів є основою для цілого набору технологій, які розробники можуть використовувати при побудові різних застосунків. Наприклад, для роботи з базами даних в цьому стеку технологій передбачена технологія ADO.NET і Entity Framework Core. Для побудови графічних застосунків з багатим інтерфейсом - технологія WPF і WinUI, для створення простих графічних застосунків - Windows Forms. Для розробки кросплатформенних мобільних і настільних застосунків - Xamarin/MAUI. Для створення веб-сайтів і веб-застосунків - ASP.NET і т.д.;
- до цього слід додати - активно розвивається і набирає популярності Blazor - фреймворк, який працює на основі .NET і дозволяє створювати веб-застосунки як на стороні сервера, так і на стороні клієнта. А в майбутньому буде підтримувати створення мобільних застосунків і, можливо, настільних застосунків;
- продуктивність - згідно з рядом тестів веб-застосунки на .NET 7 в деяких категоріях значно випереджають веб-застосунки, побудовані за допомогою інших технологій. Застосунки на .NET 7 взагалі відрізняються високою продуктивністю[18][19].

Також варто відзначити таку особливість мови C# і фреймворка .NET, як автоматичне збирання сміття. І це означає, що в більшості випадків нам не доведеться, на відміну від C++, турбуватися про звільнення пам'яті. Вищезгадане загальномовне середовище CLR само викличе збирач сміття і очистить пам'ять. Часто програму, створену на мові C#, називають керованим кодом (managed code). Що це означає? Це означає, що дана програма створена на основі платформи .NET

і, отже, керується загальномовною середовищем CLR, яке завантажує програму і, за потреби, очищує пам'ять. Однак є також програми, наприклад, створені на мові C++, які компілюються не в загальну мову CIL, як C#, VB.NET або F#, а в звичайний машинний код. У цьому випадку .NET не керує програмою. У той же час платформа .NET надає можливості для взаємодії з некерованим кодом. Код на C# компілюється в застосунки або збірки з розширеннями .exe або .dll на мові CIL (Common Intermediate Language). При запуску такого застосунку відбувається JIT-компіляція (Just-In-Time), яка перетворює його в машинний код для подальшого виконання. Оскільки наш застосунок може бути великим і містити багато інструкцій, на даний момент буде компілюватися лише та частина коду, до якої безпосередньо відбувається звернення. Якщо ми звернемося до іншої частини коду, вона буде скомпільована з CIL в машинний код. При цьому вже скомпільована частина застосунку зберігається до завершення роботи програми. В результаті цього підвищується продуктивність[20]. Після ознайомлення з можливостями мови C# та можливостями фреймворку .NET потрібно проаналізувати інструмент Xamarin та можливості для розробки програмних рішень, які він надає. Xamarin - це відкрита платформа, створена для розробки сучасних продуктивних застосунків для iOS, Android та Windows з використанням .NET. Xamarin є абстракційним рівнем, який забезпечує взаємодію між загальним кодом та кодом базової платформи. Виконання Xamarin відбувається в керованому середовищі, яке реалізує такі можливості, як управління пам'яттю та збірка сміття. Завдяки Xamarin, в середньому 90% коду застосунка може бути використано без змін на різних платформах. За допомогою цього шаблону розробник може написати весь бізнес-логіку на одній мові програмування (або використовувати існуючий код застосунка), а отримати характеристики продуктивності, оформлення та поведінки, властиві для кожної відповідної платформи. Застосунки Xamarin можна розробляти на ПК або Mac та компілювати в власні пакети

застосунків, наприклад, у файли з розширенням .apk для Android або .ipa для iOS. Платформа Xamarin розрахована на такі завдання: спільне використання коду, рішень та бізнес-логіки на різних платформах та створення застосунків мовою # у Visual Studio.

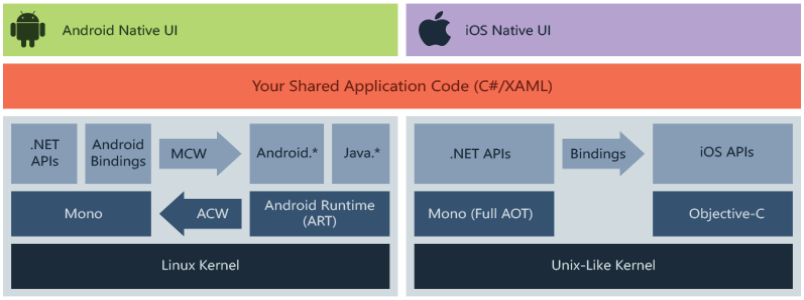


Рисунок 2.1 – загальна архітектура кросплатформеного застосунку Xamarin

За допомогою Xamarin ви можете створювати власний користувацький інтерфейс для кожної платформи та створювати загальну бізнес-логіку мовою C#, яка буде використовуватися на різних платформах. У більшості випадків Xamarin дозволяє використовувати 80% коду застосунка на різних платформах. В основі Xamarin лежить середовище .NET, яке автоматично вирішує такі завдання, як управління пам'яттю, збірка сміття та забезпечення взаємодії з базовими платформами.

Xamarin поєднує в собі можливості власних платформ з додаванням можливостей, до яких відносяться:

- повна прив'язка до базових пакетів SDK. Xamarin містить прив'язки практично для всіх базових пакетів SDK в iOS та Android. Крім того, ці прив'язки є строго типізованими, що означає, що вони зручні для навігації та використання, а також дозволяють здійснювати якісну перевірку типів під час компіляції та розробки. Строго типізовані прив'язки допомагають зменшити кількість помилок під час виконання та підвищують якість застосунків;

- взаємодія Objective-C, Java, C і C++. Xamarin дозволяє безпосередньо викликати бібліотеки Objective-C, Java, C і C++ для більш ефективного використання різноманітного стороннього коду. Ця можливість дозволяє використовувати наявні бібліотеки iOS і Android, написані на Objective-C, Java або C/C++. Крім того, Xamarin надає проекти прив'язок для прив'язки власних бібліотек Objective-C і Java за допомогою декларативного синтаксису;
- сучасні конструкції мови. Застосунки Xamarin написані на сучасною мовою C#, яка має значні покращення порівняно з Objective-C та Java. Сюди входять динамічні функції мови, функціональні конструкції, такі як лямбда-вирази, LINQ, функції паралельного програмування, універсальні шаблони тощо;
- надійна бібліотека базових класів (BCL). Застосунки Xamarin використовують бібліотеку BCL .NET, велику колекцію класів зі всебічними та спрощеними можливостями, включаючи підтримку XML, баз даних, серіалізації, введення-виведення, рядків, мережевих функцій тощо. Існуючий код C# можна компілювати для використання в застосунках, що надає доступ до тисяч бібліотек з додатковими функціями, що виходять за межі BCL;
- сучасне інтегроване середовище розробки (IDE). Xamarin використовує сучасне середовище Visual Studio, в якому реалізовані такі можливості, як автозавершення коду, вдосконалена система керування проектами і рішеннями, вичерпна бібліотека шаблонів проектів, інтегрована система керування версіями та багато іншого;
- підтримка кросплатформенних мобільних застосунків. Xamarin пропонує вдосконалену кросплатформову підтримку для трьох основних платформ -

iOS, Android і Windows. Обсяг спільного коду в створених застосунках може досягати 90%, а бібліотека Xamarin.Essentials пропонує універсальний API-інтерфейс для доступу до спільних ресурсів на всіх трьох платформах. Це дозволяє значно зменшити витрати на розробку та час випуску продуктів на ринок для розробників, які створюють мобільні застосунки.

Розглянемо особливості роботи Xamarin на кожній з доступних йому платформ[21].

Застосунки Xamarin.Android компілюються з мови C# в проміжний мовний код (IL), який під час запуску застосунка проходить Just-in-Time (JIT) компіляцію в машинний код. Застосунки Xamarin.Android працюють у середовищі виконання Mono паралельно з віртуальною машиною виконання Android (ART). Xamarin надає прив'язки .NET до просторів імен Android.* та Java.*. Середовище виконання Mono звертається до цих просторів імен за допомогою керованих викликаних оболонок (MCW) і надає середовищу виконання ART викликані програмні оболонки Android (ACW), що дозволяє обом середовищам викликати код одне одного.

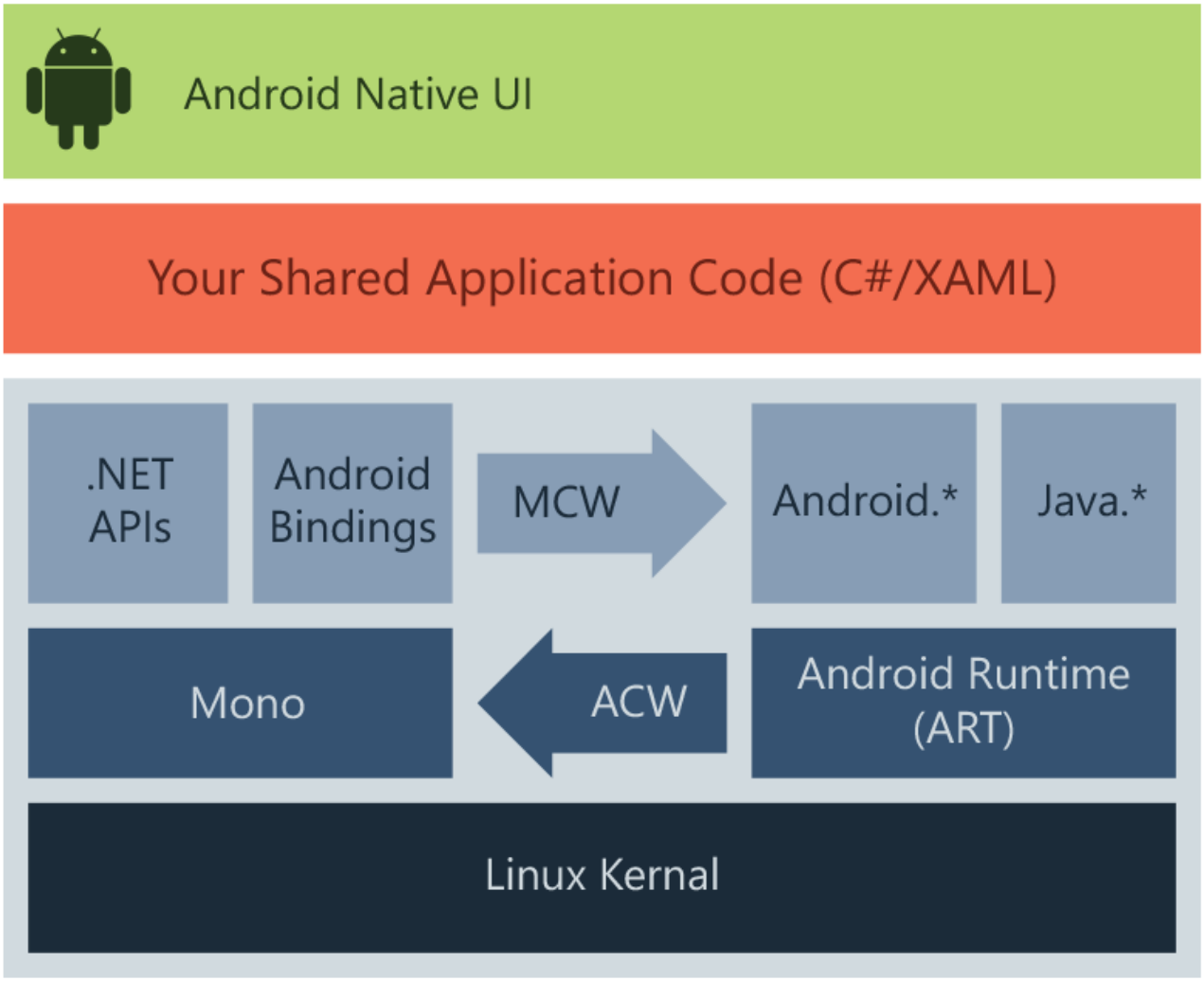


Рисунок 2.2 – Схема роботи застосунку Xamarin.Android

Застосунки Xamarin.iOS проходять повну Ahead-of-Time компіляцію (AOT) з мови C# у власний код збірки ARM. Xamarin використовує селектори для надання коду Objective-C керованому коду C# і Registrars для надання керованого коду C# коду Objective-C. Селектори і Registrars разом називаються "прив'язками" і забезпечують взаємодію між Objective-C і C#.

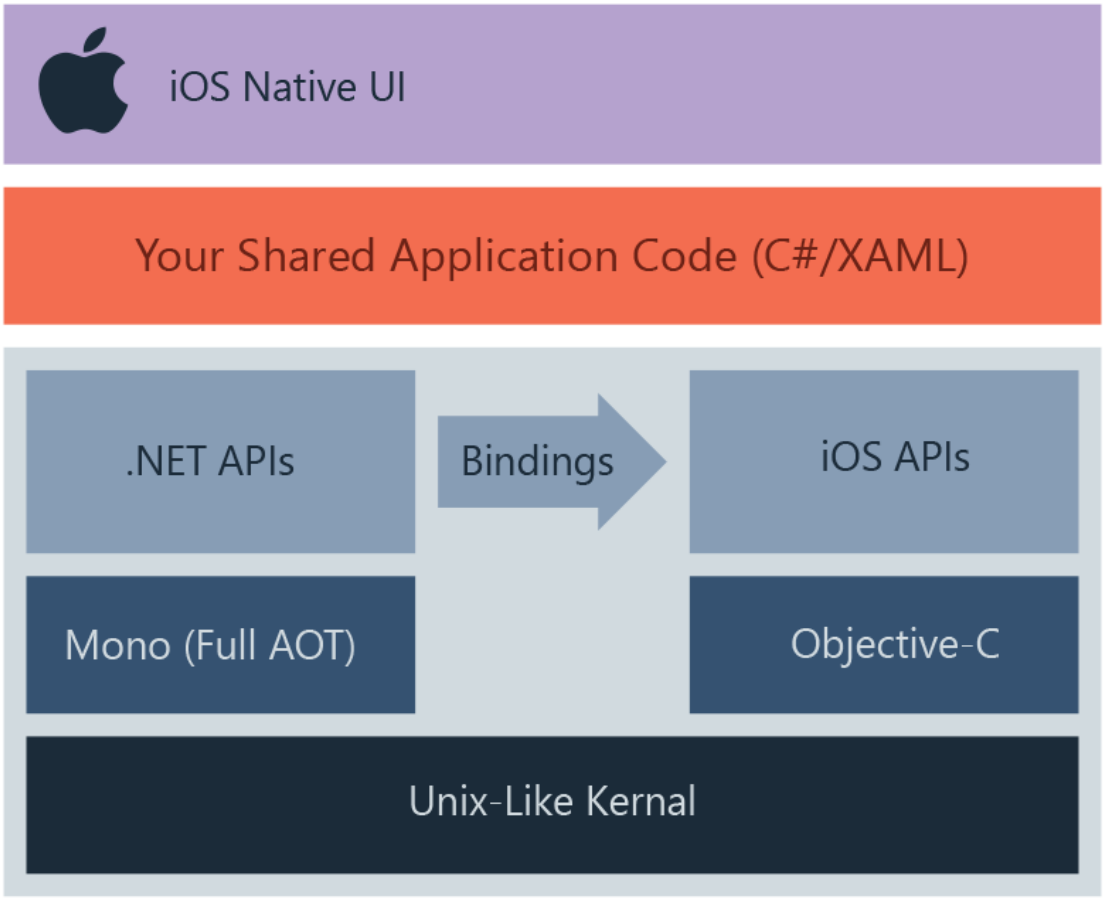


Рисунок 2.3 Схема роботи застосунку Xamarin.iOS

Xamarin.Forms - є платформою з відкритим вихідним кодом для створення користувацького інтерфейсу. За допомогою Xamarin.Forms розробники можуть створювати застосунки для Xamarin.iOS, Xamarin.Android та Windows на основі спільного коду. Xamarin.Forms дозволяє розробникам створювати користувацький інтерфейс у XAML з використанням програмного коду на C#. Ці користувацькі інтерфейси на кожній платформі підготовляються для перегляду як власні елементи керування. Нижче наведено деякі приклади функцій, які надає Xamarin.Forms:

- мова користувацького інтерфейсу XAML;
- прив'язка даних;

- жести;
- візуальні ефекти;
- налаштування стилів.

Ось кілька прикладів популярних застосунків, які були створені з використанням Xamarin:

1) Slack: Slack є популярною платформою спілкування для команд та організацій. Застосунок Slack був розроблений з використанням Xamarin, що дозволяє користувачам обмінюватися повідомленнями, документами та інформацією у режимі реального часу.

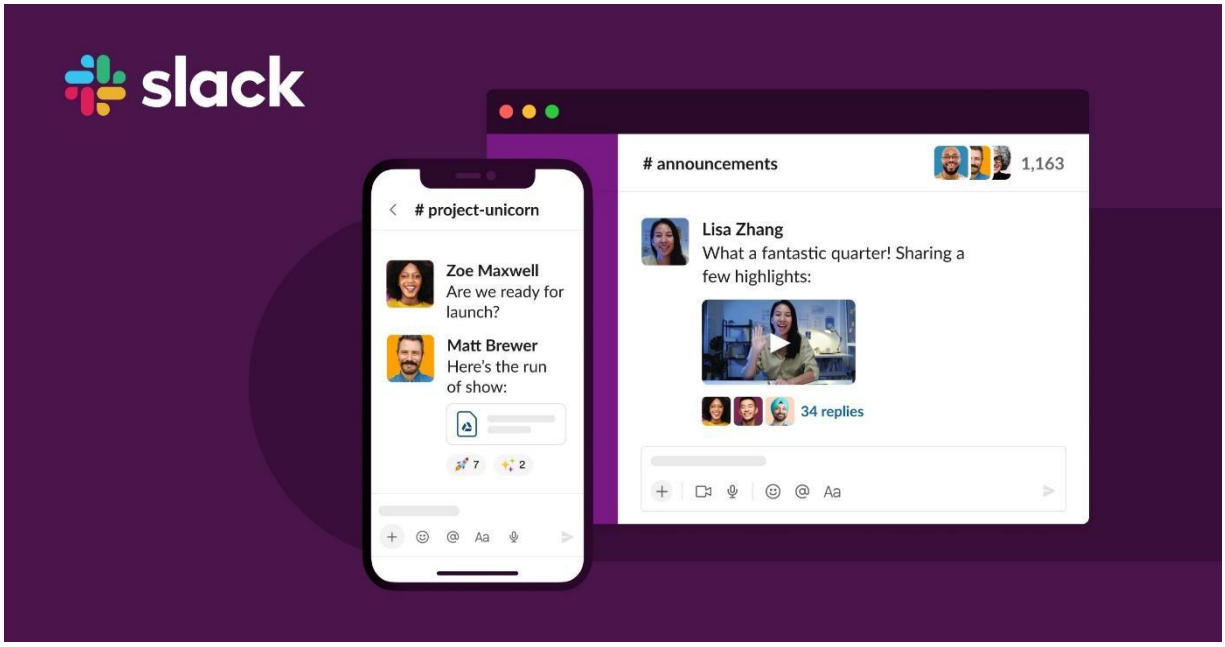


Рисунок 2. – Інтерфейс застосунку Slack, створений з використанням Xamarin

2) Pinterest: Pinterest є соціальною мережею та платформою для зберігання та пошуку інтересних зображень та ідей. Застосунок Pinterest був розроблений з використанням Xamarin, що дозволяє користувачам зберігати, організовувати та ділитися зображеннями та контентом.

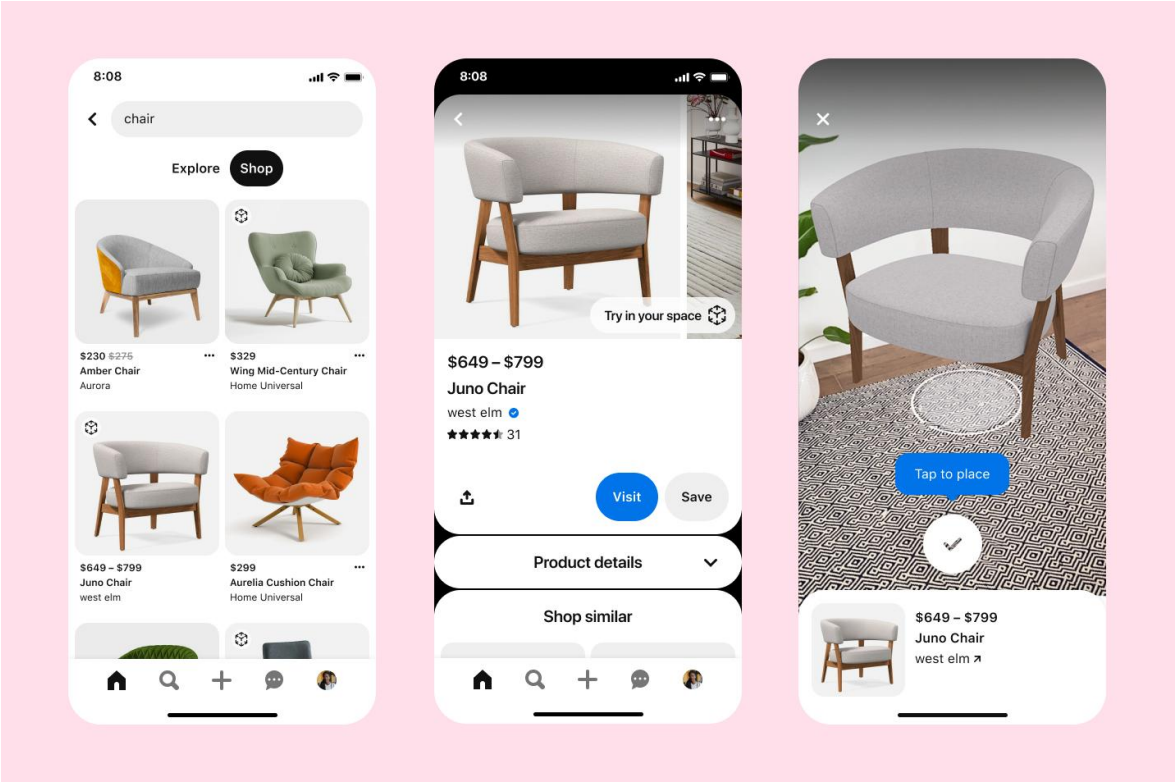


Рисунок 2. – Інтерфейс застосунку Pinterest з використанням Xamarin

2.2 Серверне рішення для мобільного застосунку

Xamarin є фреймворком для розробки мобільних клієнтських застосунків, але для створення серверної частини застосунку на Xamarin потрібно використовувати додаткові інструменти та технології.Через те, що для архітектури мобільного застосунку була обрана архітектура BaaS(бекенд як сервер) потрібно обрати платформу для створення серверної частини застосунку. Є декілька популярних варіантів для реалізації такого програмного рішення:

1. ASP.NET Web API: Використання ASP.NET Web API дозволяє створити RESTful веб-сервіси, які можуть бути використані в мобільному застосунку Xamarin. ASP.NET Web API надає потужні можливості для обробки запитів, маршрутизації, аутентифікації та авторизації, обміну даними в форматі JSON або XML[22];

2. Node.js з Express.js: Використання Node.js разом з фреймворком Express.js дозволяє створювати серверні застосунки з високою продуктивністю на базі JavaScript. Node.js має потужну екосистему пакетів, що дозволяє легко створювати серверну логіку, обробляти запити, працювати з базами даних та іншими операціями;
3. Firebase: Firebase є платформою Backend-as-a-Service (BaaS), яка надає широкий набір інструментів для розробки серверної частини. Firebase пропонує такі можливості, як аутентифікація користувачів, реальний час бази даних, хмарне сховище, пуш-сповіщення та багато іншого;
4. .NET Core: .NET Core є відкритою, крос-платформеною версією фреймворку .NET, яка дозволяє створювати серверні застосунки для різних операційних систем, включаючи Windows, macOS та Linux. Ми можемо використовувати .NET Core для створення серверної логіки вашого застосунку Xamarin, використовуючи ASP.NET Core фреймворк для створення веб-сервісів.

Для свого проекту я обрав платформу Google Firebase через ряд причин, які будуть вказані далі.

Firebase - це платформа Backend-as-a-Service (BaaS), розроблена Google, яка надає різноманітні послуги та інструменти для розробки мобільних застосунків, включаючи ті, які розроблені на Xamarin. Ось докладний опис деяких послуг, які надає Firebase для розробки мобільних застосунків:

- Firebase Authentication: Ця послуга дозволяє легко впроваджувати систему аутентифікації для вашого мобільного застосунку. Вона підтримує автентифікацію через електронну пошту та пароль, облікові записи Google, Facebook, Twitter та інші провайдери. Firebase Authentication забезпечує безпеку та надійність аутентифікаційного процесу;

- Cloud Firestore: Це розподілена база даних, яка забезпечує зручне зберігання та синхронізацію даних між клієнтами та сервером. Cloud Firestore підтримує реальний час оновлення даних, запити до бази даних, відслідковування змін та інші функції, що спрощують роботу з даними мобільного застосунку;
- Cloud Storage: Ця послуга надає можливість зберігати та керувати медіафайлами (зображення, відео, аудіо тощо) у хмарному сховищі. Ви можете завантажувати файли до Cloud Storage, отримувати прямі посилання на файли, керувати правами доступу та іншими параметрами;
- Firebase Cloud Messaging (FCM): Ця послуга дозволяє надсилати пуш-сповіщення на мобільні пристрої. Можна використовувати FCM для надсилання сповіщень користувачам з сервера або з консолі Firebase. Це може бути корисно для повідомлень про нові повідомлення, сповіщень або оновлення в застосунку;
- Firebase Analytics: Цей інструмент дозволяє збирати, аналізувати та відстежувати дані про використання мобільного застосунку. Ви можете отримувати статистику про активність користувачів, конверсію, поведінку користувачів та інші важливі метрики. Це допомагає розуміти, як користувачі взаємодіють з застосунком та покращувати його функціональність[23][24][25][26].

У порівняння з іншими варіантами, Firebase – готовий набір інструментів для розміщення мобільного застосунку. Для того щоб інтегрувати запропоновану серверну частину в свій проект потрібно створити сторінку застосунку та обрати інструменти для вашого мобільного застосунку. Firebase надає безкоштовний тестовий проект з певними лімітами на користування на день\місяць[27].

1. Обмежена кількість активних користувачів: Тестовий проект Firebase може мати обмежену кількість активних користувачів, які можуть використовувати застосунок одночасно.
2. Обмеження трафіку: Firebase може мати обмеження на обсяг трафіку, який може переходити через його сервери для тестового проекту. Це може впливати на розмір файлів, передачу даних та інші функції, що потребують мережевого з'єднання.
3. Обмежена кількість баз даних та документів: Тестовий проект може мати обмежену кількість баз даних або обмежену кількість документів, які можна зберігати в Firebase. Це може впливати на обсяг даних, які можна зберігати та обробляти в застосунку.
4. Обмеження на кількість повідомлень або сповіщень: Firebase може мати обмеження на кількість повідомлень або сповіщень, які можна надіслати за певний період часу. Це може впливати на можливості розсилки повідомлень або сповіщень користувачам.

Основні переваги платформи Firebase в тому, що вона є дуже гнучким інструментом для налаштування, моніторингу та керування проектом, які було б складніше та більш дорожче реалізувати самостійно:

1. Authentication Providers: Firebase підтримує різноманітні провайдери аутентифікації, такі як Google, Facebook, Twitter, GitHub та інші. Можливо додати можливість входу через ці провайдери в свій мобільний застосунок, що полегшить користувачам процес реєстрації та входу;
2. Real-time Database: Firebase має реальний час бази даних, що дозволяє синхронізувати дані між клієнтами і сервером в реальному часі. Є

можливість використовувати цю базу даних для зберігання і отримання даних, які автоматично оновлюються на всіх підключених пристроях;

3. Cloud Functions: Ця функція дозволяє створювати та розгортати серверні функції, які можуть реагувати на події в застосунку або в хмарному середовищі Firebase. Можна використовувати ці функції для обробки бізнес-логіки, виконання операцій з базою даних, відправки повідомлень та багато іншого;
4. Cloud Messaging: Firebase Cloud Messaging (FCM) дозволяє надсилати пуш-сповіщення на мобільні пристрої. Є можливість надсилати сповіщення користувачам на основі різних подій, таких як нові повідомлення, оновлення застосунку, акції тощо;
5. Remote Config: Ця послуга дозволяє вам змінювати поведінку вашого застосунку на віддаленому сервері без необхідності оновлення застосунку. Ви можете налаштувати різні параметри, такі як зовнішній вигляд, функціональність та інші, і змінювати їх у реальному часі;
6. Performance Monitoring: Firebase надає інструменти для моніторингу продуктивності застосунку. Можна час завантаження сторінок, кількість помилок, використання пам'яті та інші показники для виявлення проблем та оптимізації продуктивності;
7. Test Lab: Firebase Test Lab дозволяє автоматизовано тестувати застосунок на різних пристроях та конфігураціях. Можливо запустити різні тести, включаючи функціональне тестування, тестування витривалості, тестування на реальних пристроях та інші[28].

Підсумувавши все, до причин того, що я обрав саме Firebase для виконання бакалаврського завдання можна додати 1) широкий спектр функціональності:

Firebase надає різноманітні сервіси, такі як аутентифікація користувачів, база даних в реальному часі, зберігання файлів, хостинг веб-сайтів та багато інших; 2) Інтеграція з Xamarin: Firebase надає набір SDK для Xamarin, що спрощує процес інтеграції Firebase сервісів до мобільного застосунку Xamarin[29][30]. Це може включати в себе наявність готових компонентів, документацію та приклади коду, що допомагають швидко і ефективно використовувати Firebase у своєму проекті; 3) Firebase пропонує розширені можливості аналітики, що дозволяють отримувати дані про використання застосунку, поведінку користувачів та ефективність функціоналу; 4) Firebase базується на інфраструктурі Google, що забезпечує масштабованість та надійність сервісів. Можна бути впевненим, що мобільний застосунок буде масштабуватись, коли кількість користувачів зростатиме, і Firebase забезпечить стабільну роботу сервісів. У моєму випадку аудиторія навряд-чи буде збільшуватись, проте великою перевагою для використання саме Firebase є її висока надійність як частини інфраструктури Google та стабільність надання послуг.

Висновки до розділу

Розробка сучасного мобільного застосунку нараховує велику кількість етапів. В першу чергу потрібно було проаналізувати інструмент для виконання бакалаврської роботи – Xamarin, визначити його можливості та особливості роботи з ним. Після цього потрібно було обрати серверну частину для розробки мобільного застосунку. Було визначено, що для цієї ролі підійде Firebase через ряд наведених характеристик цієї платформи та наведені переваги перед іншими трьома варіантами. Були виділені характеристики мови програмування та інструментів для виконання бакалаврської роботи. Була обрана архітектура побудови програмного застосунку.

РОЗДІЛ 3

ПРОЕКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКА ЧАТУ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ XAMARIN

3.1 Опис розроблюваної системи

3.1.1 Інтерфейс користувача

Одним з найскладніших завдань про розробці програмних рішень у сфері мобільних застосунків є розробка інтерфейсу користувача, або User Interface (від англ.). Користувацький інтерфейс це інструмент, завдяки якому користувач через клієнтську сторону застосунку взаємодіє з серверною частиною застосунку, що, в свою чергу, є взаємодією з іншими користувачами у випадку користування застосунком-чатом, як у нашому випадку. Основне завдання користувацького інтерфейсу для кожного застосунку (не тільки мобільного) бути інтуітивно зрозумілим для користувача та мати приємний зовнішній вигляд (палітра, стиль тощо). Для досягнення цієї мети існує велика кількість інструментів у мережі Інтернет.

Одним з першочергових завдань при розробці користувацького інтерфейсу є вибір кольорової палітри застосунка. Для допомогу у генерації та виборі кольорової палітри був залучений сервіс Adobe, який дозволяє обрати палітру з вже існуючих чи створити власну палітру. Платформа надає можливість дізнатися сучасні тенденції у сфері розробки кольорових схем у сучасних застосунках, отримати доступ до великої кількості вже сформованих бібліотек та палітр або створити власну за допомогою великого обсягу наданих інструментів. Є можливість обрати основний базовий колір, на основі якого вбудовані інструменти платформи запропонують набір кольорів для того щоб палітра виглядала гармонічно з точки зору кольорів[31].

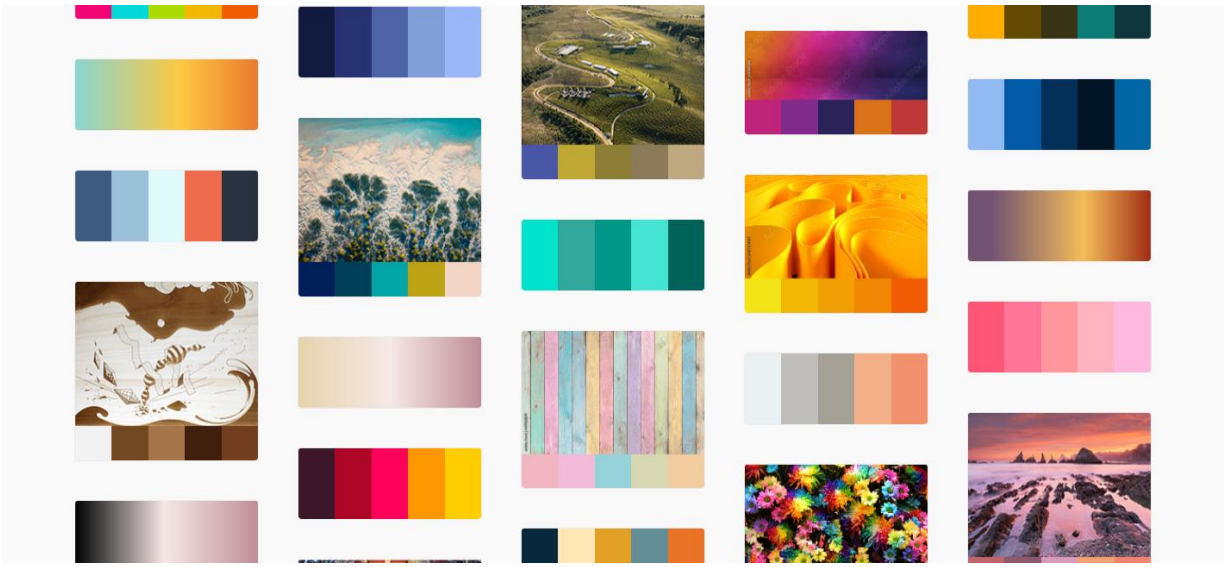


Рисунок 3.1 – Можливий вибір кольорових палітр на сервісі Adobe

Для свого застосунку я вирішив створити власну палітру кольорів:

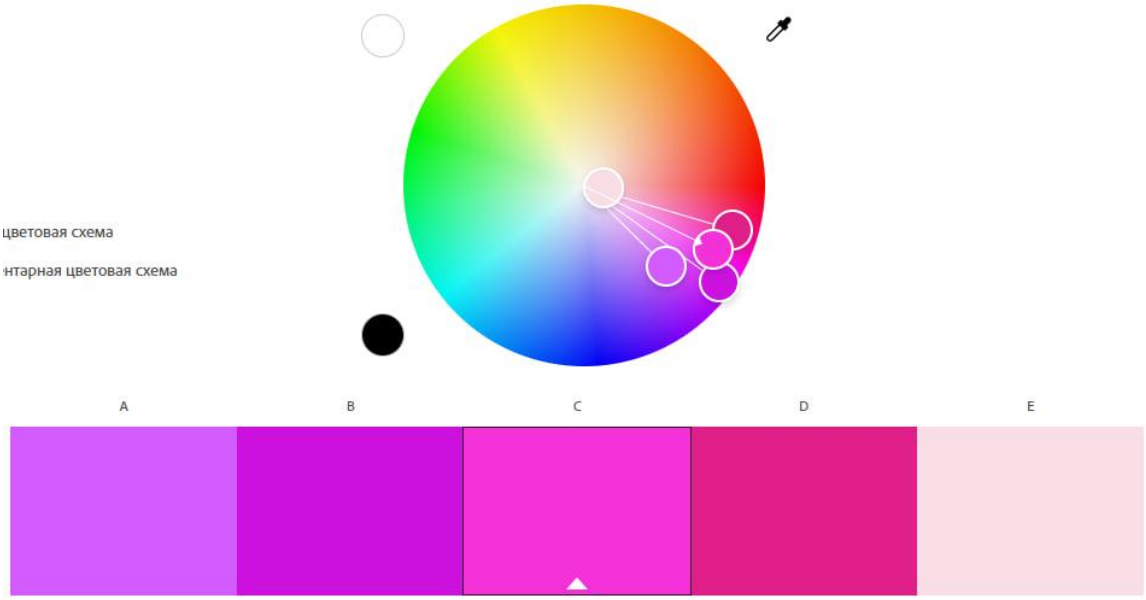


Рисунок 3.2 – власноруч створена палітра кольорів для застосунку

Оскільки за завданням бакалаврської роботи має бути використаний фреймворк Xamarin, то саме він відповідає за розробку користувацького інтерфейсу мобільного застосунку. Xamarin надає широкий список інструментів для модифікації користувацького інтерфейсу.

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="launcher_background">#FFFFFF</color>
    <color name="colorPrimary">#9c27b0</color>
    <color name="colorPrimaryDark">#9c27b0</color>
    <color name="colorAccent">#9c27b0</color>
    <color name="splash_background">#ffffff</color>
</resources>
```

Рисунок 3.3 – імплементація основного кольору в застосунок

Xamarin.Forms потребує деякої кількості властивостей для різних кольорів, до яких будуть звертатися компоненти застосунку, наприклад:

- основний колір – використовується для основних елементів інтерфейсу користувача, є найбільш поширеним у застосунку;
- другорядний колір – не є обов'язковим, але рекомендується до використання для того щоб акцентувати увагу користувача на собі та для ліпшої навігації у застосунку;
- текстовий колір – використовується для тексту у всьому застосунку, може бути локально змінений у кожному компоненті.

3.1.2 Структура мобільного застосунку

Проаналізувавши популярні месенджери на ринку, можна отримати представлення про структуру якісного мобільного застосунку. Для репрезентації цього бачення була створена схема структури мобільного застосунку для бакалаврської роботи:

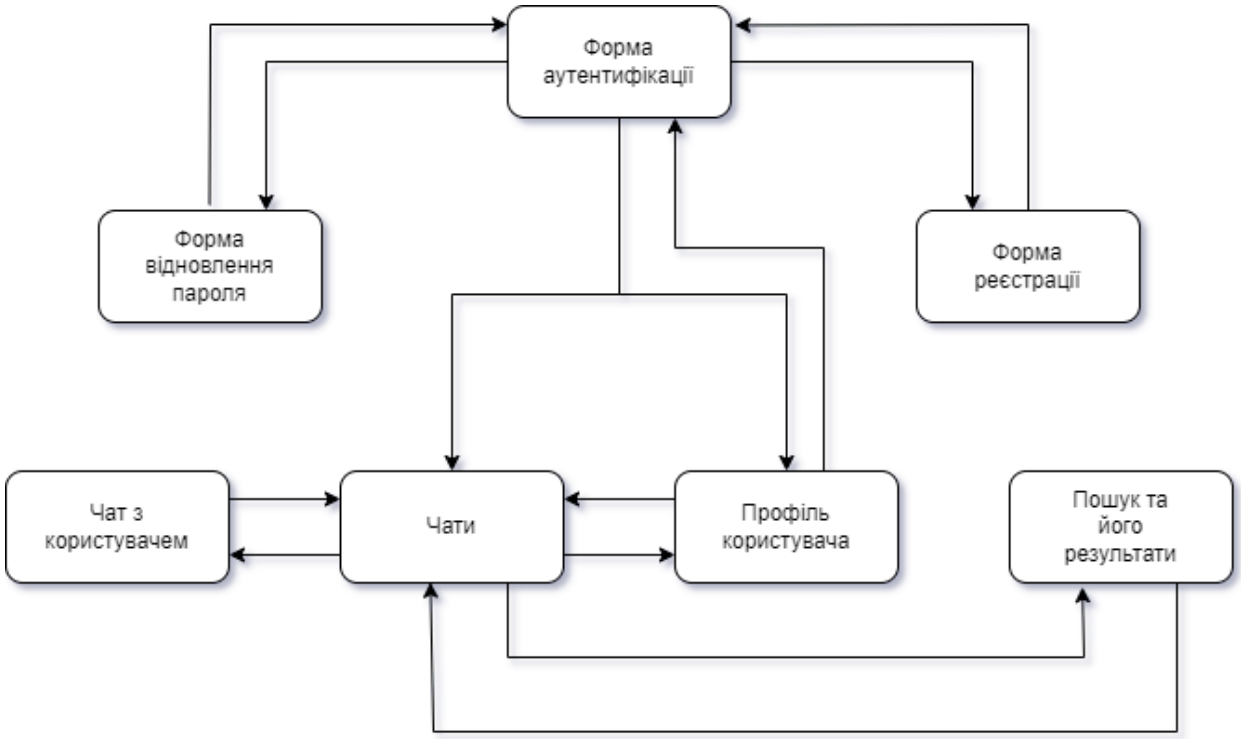


Рисунок 3.4 – діаграма структури застосунку

3.2 Процесс створення застосунку з поданої структури

3.2.1 Аутентифікація

Авторизація у застосунок відбувається за допомогою сервісу аутентифікації Firebase, яка відбувається таким чином:

- інтеграція: необхідно підключити бібліотеку Firebase Authentication до мобільного застосунку на платформі Xamarin. Це забезпечує доступ до функцій авторизації Firebase;

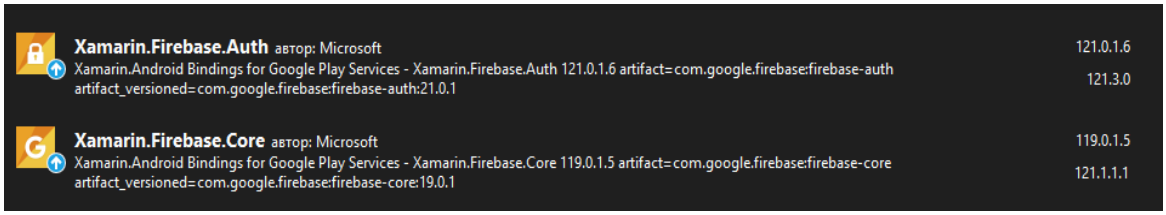


Рисунок 3.5 – підключення бібліотек Firebase для авторизації за допомогою менеджера пакетів NuGet

- налаштування провайдерів авторизації: налаштування можливої авторизація за допомогою різних провайдерів, таких як електронна пошта, Google, та інше. У випадку мого застосунку налаштована аутентифікація за допомогою електронної пошти;
- аутентифікація користувача: Коли користувач намагається увійти в застосунок, використовуючи обраний метод авторизації, застосунок викликає відповідний метод з Firebase SDK;
- перевірка авторизації: Firebase Authentication здійснює перевірку переданих даних та зв'язок з відповідним провайдером авторизації (наприклад, Google або Facebook) для перевірки дійсності авторизаційних даних;
- результат авторизації: Після успішної перевірки, Firebase надає зворотний виклик до застосунку з даними про авторизованого користувача, такі як ідентифікатор, ім'я, електронна пошта тощо. Застосунок може використовувати ці дані для подальшої роботи з авторизованим користувачем;
- управління користувачами: Firebase також надає можливості управління користувачами, такі як створення нових облікових записів, зміна паролів, відновлення доступу до облікового запису тощо;

Authentication Providers: Firebase підтримує різноманітні провайдери аутентифікації, такі як Google, Facebook, Twitter, GitHub та інші. Можливо додати можливість входу через ці провайдери в свій мобільний застосунок, що полегшить користувачам процес реєстрації та входу;

```
Ссылок: 1
public class FirebaseAuthService : IFirebaseAuth
{

    DataClass dataClass = DataClass.GetInstance;

    Ссылок: 1
    public FirebaseAuthResponseModel IsLoggedIn()
    {
        try
        {
            if (FirebaseAuth.Instance.CurrentUser.Uid == null)
            {
                dataClass.isSignedIn = false;
                dataClass.loggedInUser = new UserModel();
                return new FirebaseAuthResponseModel() { Status = false, Response = "Currently logged out." };
            }

            dataClass.loggedInUser = new UserModel()
            {
                uid = FirebaseAuth.Instance.CurrentUser.Uid,
                email = FirebaseAuth.Instance.CurrentUser.Email,
                username = dataClass.loggedInUser.username,
                userType = dataClass.loggedInUser.userType,
                contacts = dataClass.loggedInUser.contacts,
                createdAt = dataClass.loggedInUser.createdAt
            };
            dataClass.isSignedIn = true;
            return new FirebaseAuthResponseModel() { Status = true, Response = "Currently logged in." };
        }
        catch (Exception ex)
        {
            dataClass.isSignedIn = false;
            dataClass.loggedInUser = new UserModel();
            return new FirebaseAuthResponseModel() { Status = false, Response = ex.Message };
        }
    }
```

Рисунок 3.6 – фрагмент класу авторизації, який зв'язується з базою даних та відповідає за перевірку даних користувача, який намагається увійти до системи

За допомогою інструментів Xamarin був створений користувацький інтерфейс сторінки для аутентифікації:

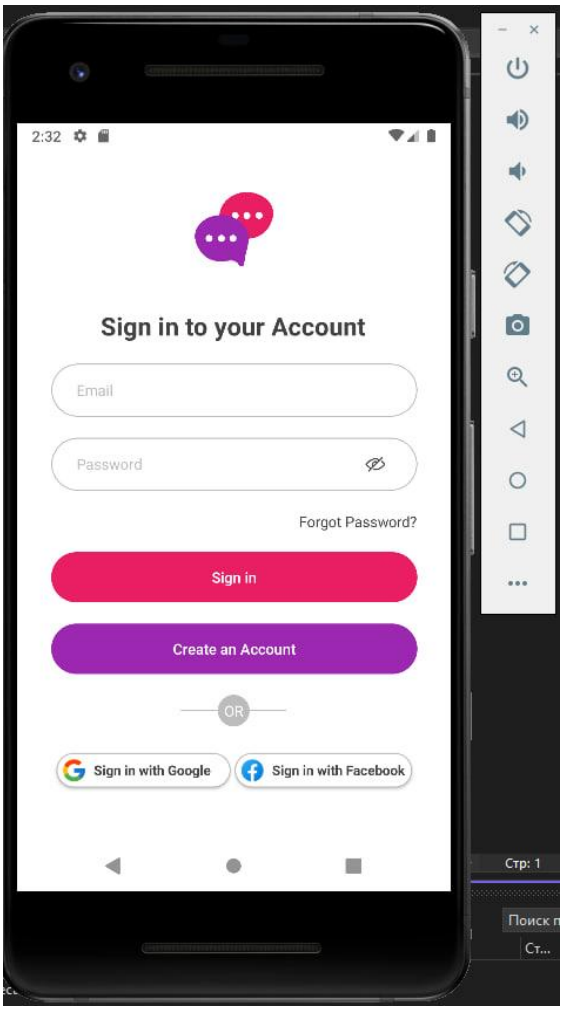


Рисунок 3.7 Форма для аутентифікації користувачів у систему.

3.2.2 Обмін з базою даних

Друга важлива частина – обмін з базою даних для відправлення та отримання повідомлень користувачів. Firebase надає серверне рішення для цієї проблеми у вигляді модуля Firebase Cloud Messaging (FCM).

Рисунок 3.8 – Бібліотека CloudFirestore, яка дозволяє отримати доступ до інструментів для роботи з бекенд-частиною, наданою Firebase.

Firebase Cloud Messaging (FCM) є сервісом повідомлень, який надає Firebase для надсилання повідомлень на мобільні пристрої. Він забезпечує надійну та масштабовану доставку повідомлень до застосунків на пристроях Android, iOS та веб-браузерах.

Основні компоненти та принципи роботи Firebase Cloud Messaging:

- клієнтська сторона: застосунок на мобільному пристрої або веб-браузері, який отримує повідомлення від сервісу FCM. Клієнтська сторона повинна бути налаштована для отримання токена реєстрації, який ідентифікує пристрій або браузер у системі;
- серверна сторона: сервер, який відправляє повідомлення до FCM за допомогою API. Серверна сторона може відправляти повідомлення на один або кілька пристроїв, використовуючи токени реєстрації або теми повідомлень;
- Firebase Cloud Messaging: централізований сервіс, який приймає запити від серверної сторони та доставляє повідомлення на пристрої або браузери.

Процес роботи Firebase Cloud Messaging:

- реєстрація пристрою: При запуску застосунку на пристрої, він отримує унікальний токен реєстрації від сервісу FCM. Цей токен використовується для ідентифікації пристрою при відправці повідомлень.
- відправка повідомлень: Серверна сторона відправляє запит до сервісу FCM з вказанням токенів реєстрації або тем повідомлень, а також змісту повідомлення. FCM перевіряє валідність токенів та доставляє повідомлення до відповідних пристроїв або браузерів;

- доставка повідомлень: Firebase Cloud Messaging забезпечує надійну доставку повідомлень до пристроїв, навіть у разі, якщо пристрій тимчасово відключений або не активний. Повідомлення може бути доставлено за допомогою пуш-повідомлень або при наступному активуванні застосунку;
- обробка повідомлень на клієнтській стороні: Застосунок на пристрої або браузері отримує повідомлення від FCM і може обробити його відповідно до власної логіки. Це може включати відображення сповіщень, оновлення інтерфейсу або виконання певних дій.

```
Ссылок: 7
public class ConversationModel : INotifyPropertyChanged
{
    Ссылок: 2
    string _id { get; set; }
    Ссылок: 2
    string _message { get; set; }
    Ссылок: 2
    string _converseeID { get; set; }
    Ссылок: 2
    DateTimeOffset _createdAt { get; set; }

    Ссылок: 11
    public string id { get { return _id; } set { _id = value; OnPropertyChanged(nameof(id)); } }
    Ссылок: 2
    public string message { get { return _message; } set { _message = value; OnPropertyChanged(nameof(message)); } }
    Ссылок: 3
    public string converseeID { get { return _converseeID; } set { _converseeID = value; OnPropertyChanged(nameof(converseeID)); } }
    Ссылок: 2
    public DateTimeOffset createdAt { get { return _createdAt; } set { _createdAt = value; OnPropertyChanged(nameof(createdAt)); } }

    public event PropertyChangedEventHandler PropertyChanged;
    Ссылок: 4
    protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = "")
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Рисунок 3.9 – клас для обробки повідомлень користувачів

```
namespace ChatApp
{
    Ссылок: 30
    public class FirebaseAuthResponseModel : INotifyPropertyChanged
    {
        Ссылок: 2
        bool _Status { get; set; }
        Ссылок: 2
        string _Response { get; set; }

        Ссылок: 17
        public bool Status { get { return _Status; } set { _Status = value; OnPropertyChanged(nameof(Status)); } }
        Ссылок: 19
        public string Response { get { return _Response; } set { _Response = value; OnPropertyChanged(nameof(Response)); } }

        public event PropertyChangedEventHandler PropertyChanged;
        Ссылок: 2
        protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = "")
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
```

Рисунок 3.10 – клас для обробки зв’язку з базою даних

3.3 Огляд інтерфейсу та роботи застосунку

Розроблений програмний застосунок є мобільним мессенджером із застосуванням технології Xamarin та платформи Firebase. Інтерфейс програми сформований з: сторінки аутентифікації, сторінки відновлення пароля, сторінки реєстрації, головної сторінки з доступними чатами, сторінки пошуку друзів для спілкування та сторінки профілю.

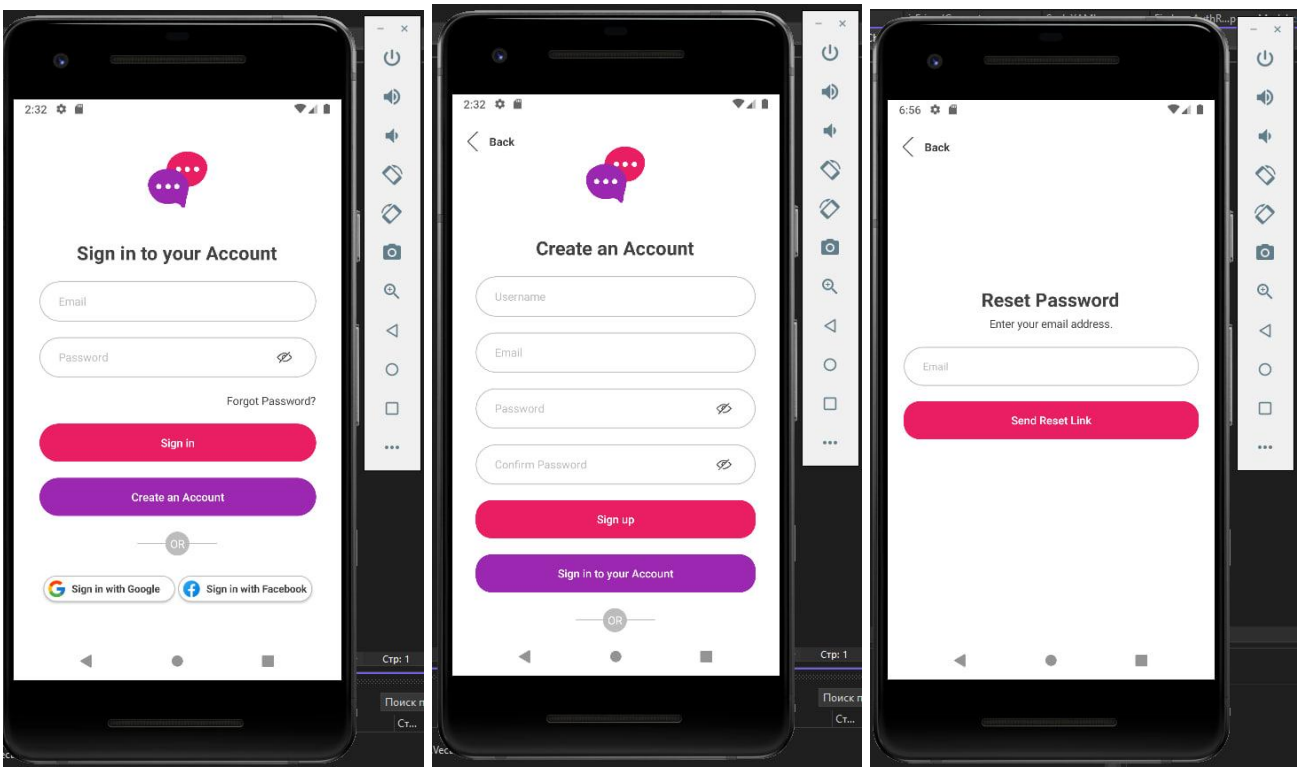
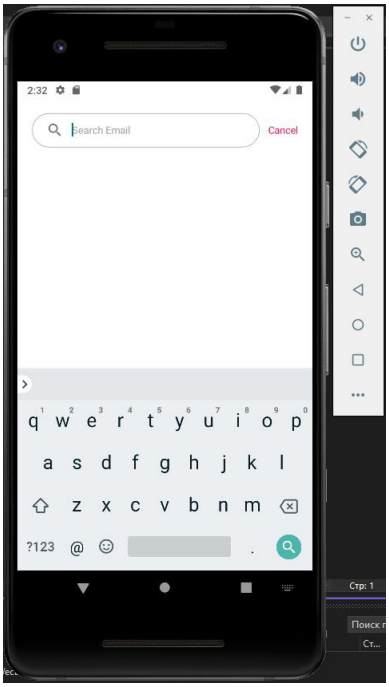


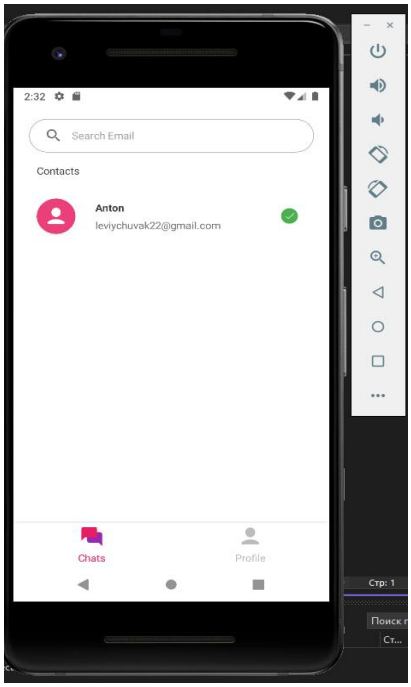
Рисунок 3.11 Сторінки для входу, створення аккаунта та для відновлення пароля

Серверні інструменти Firebase дають можливість налаштувати реєстрацію за електронною поштою та платформа надає можливість серверу надсилати на вказану пошту повідомлення для підтвердження реєстрації у застосунку. Те ж саме відбувається для відновлення паролю – на пошту сервісами Google Firebase надсилається повідомлення про відновлення паролю.

Рисунок 3.12 Приклад повідомлення на пошті

Після аутентифікації користувач потрапляє на головну сторінку, звідки він має доступ до всього функціоналу застосунку – чатів з іншими користувачами, доступ

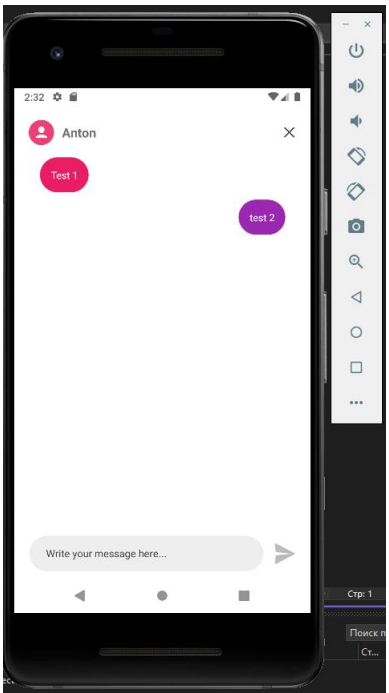


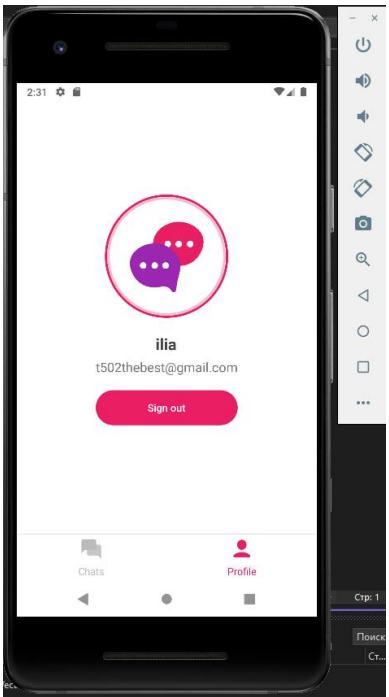


до сторінки пошуку інших користувачів та доступ до свого профілю.

Рисунок 3.13 – Інтерфейс сторінки для пошуку користувачів,інтерфейс сторінки профілю, інтерфейс головної сторінки з чатам та інтерфейс чату з користувачем

Реалізована можливість вийти з аккаунту через сторінку «Профіль» для





подальшої аутентифікації під іншим акаунтом.

Висновки до розділу

Інтерфейс був розроблений відповідно до сучасного рівня вимог до таких інтерфейсів у сфері мобільних застосунків та визначених завдань на розробку. Ціллю було розробити інтуїтивно-зрозумілий та дружній до користувача інтерфейс. Застосунок включає в себе клієнтську частину та серверну частину з інтегрованою базою даних. Відповідно до завдання бакалаврської роботи та обраних технологій була розроблена кожна з частин застосунку. Застосунок був побудований згідно з наведеними завданнями для розробки, надані зображення інтерфейсу застосунку та його функціоналу

ВИСНОВКИ

Була наведена та опрацьована статистика та тенденції розподілу інтернет-трафіку у світі за останні роки, статистика про вартість мобільних пристроїв з доступом до мережі Інтернет, були порівнянні характеристики таких пристроїв різних років, була наведена сучасна статистика користування популярними мессенджерами у світі, порівнянні їх показники.

Було визначено що Інтернет-трафік все більш «мігрує» до мобільних пристроїв, що кількість користувачів таких пристроїв збільшується с кожним роком та були проаналізовані причини цього явища – зниження середньої ціні на смартфони та суттєве покращення їх характеристик, що робить відношення ціна/якість більш привабливими для покупців з кожним роком. Були проаналізовані причини зростання популярності мобільних застосунків для спілкування – стабільне збільшення користувачів, дешевизна спілкування порівняно зі стільниковим та наземним зв’язком, швидкість зв’язку та можливість бути на зв’язку майже з будь-якої точки у будь-який час. Були проаналізовані статистичні дані про популярні месенджери у світі, на основі чого були зроблені висновки про активність аудиторії та стабільне зростання кількості користувачів. Враховуючи це, вважаю мету роботи - розробку власного програмного рішення у сфері мобільного спілкування актуальною.

Був проведений аналіз підходів до розробки застосунків-чатів та архітектурних рішень, на основі чого була обрана архітектура BaaS та платформа Firebase у доповнення до фреймворку Xamarin. Були надані характеристики цих інструментів та їх альтернатив, надані характеристики мови програмування. На основі поданих даних та наведених переваг був обраний додатковий інструмент розробки.

Було реалізоване програмне рішення у сфері мобільного спілкування відповідно до наданого завдання, були надані фрагменти коду ключового функціоналу застосунку та приклади інтерфейсу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What are Arm processor. [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/whatis/definition/ARM-processor.(date of access: 05.05.2023)
2. IDC. Smartphone Market Share [Електронний ресурс] – Режим доступу до ресурсу: https://www.idc.com/promo/smartphone-market-share.(date of access: 12.03.2023)
3. Statinvestor. Average selling price of smartphones worldwide 2016-2021 [Електронний ресурс] – Режим доступу до ресурсу: https://statinvestor.com/data/35666/smartphone-average-selling-price-worldwide/
4. Datareportal. Internet use over time. [Електронний ресурс]Режим доступу до ресурсу: https://datareportal.com/global-digital-overview#:~:text=There%20are%205.48%20billion%20unique,of%202.9%20percent%20per%20year.(date of access: 11.03.2023)
5. SimilarWeb. Most Popular Messaging Apps Worldwide 2023. [Електронний ресурс] – Режим доступу до ресурсу: https://www.similarweb.com/blog/research/market-research/worldwide-messaging-apps/.(date of access: 12.04.2023)
6. Core.Telegram. MTProto mobile protocol. [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/mtproto.(date of access: 11.04.2023)
7. An overview of HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.(date of access: 12.02.2023)
8. Open protocol development [Електронний ресурс] – Режим доступу до ресурсу: https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:developing-open-

protocols/a/open-standards-and-protocols#:~:text=An%20open%20(nonproprietary)%20protocol%20is,to%20join%20the%20global%20network. (date of access: 20.05.2023)

9. Mobile App Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.netsolutions.com/insights/mobile-app-architecture-guide/ . (date of access: 19.05.2023)
10. React Native Cross-Platform Development [Електронний ресурс] – Режим доступу до ресурсу: https://reactnative.dev/ .(date of access: 05.04.2023)
11. Client-Server Architecture Explained [Електронний ресурс] – Режим доступу до ресурсу: https://www.simplilearn.com/what-is-client-server-architecture-article#:~:text=The%20client%2Dserver%20architecture%20refers,model%20or%20client%20server%20network . (date of access: 05.04.2023)
12. Peer to Peer(P2P) Architecture [Електронний ресурс] – Режим доступу до ресурсу: https://www.enjoyalgorithms.com/blog/peer-to-peer-networks. (date of access: 05.04.2023)
13. Mobile app client architecture: The difference between native vs. hybrid vs. web apps [Електронний ресурс] – Режим доступу до ресурсу: https://www.westmonroe.com/perspectives/in-brief/mobile-client-architecture-native-vs-hybrid-vs-web-apps . (date of access: 01.04.2023)
14. What is BaaS? [Електронний ресурс] – Режим доступу до ресурсу: https://www.cloudflare.com/learning/serverless/glossary/backend-as-a-service-baas .(date of access: 20.03.2023)
15. Що таке Xamarin? [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/ru-ru/xamarin/get-started/what-is-xamarin .(date of access: 20.02.2023)

16. A tour of C# [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (date of access: 17.04.2023)
17. What's new in C# 11 [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-11.(date of access: 20.02.2023)
18. .NET [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/en-us/.(date of access: 20.02.2023)
19. What is DotNet [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/ru/what-is/net/.(date of access: 20.02.2023)
20. Full Guide C# [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/tutorial/.(date of access: 20.02.2023)
21. What is Xamarin [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/ru-ru/xamarin/get-started/what-is-xamarin.(date of access: 12.02.2023)
22. ASP.Net Get Started [Електронний ресурс] – Режим доступу до ресурсу: https://dotnet.microsoft.com/en-us/learn/aspnet/hello-world-tutorial/intro.(date of access: 05.04.2023)
23. ExpressJS web framework [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs.(date of access: 05.04.2023)
24. Firebase|Firestore [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/firestore?hl=en.(date of access: 12.03.2023)
25. Firebase Authentification [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/auth?hl=en.(date of access: 12.03.2023)

26. Cloud Firestore [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/firestore?hl=en.(date of access: 12.03.2023)
27. Firebase Cloud Messaging [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/cloud-messaging?hl=ru.(date of access: 12.03.2023)
28. Usage and limits [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/firestore/quotas?hl=ru.(date of access: 12.03.2023)
29. Що таке Firebase? [Електронний ресурс] – Режим доступу до ресурсу: https://avada-media.ua/ua/services/firebase/.(date of access: 05.05.2023)
30. SDK Firebase [Електронний ресурс] – Режим доступу до ресурсу: https://support.google.com/admob/answer/6360054?hl=uk.(date of access: 10.03.2023)
31. Color.Adobe [Електронний ресурс] – Режим доступу до ресурсу: https://color.adobe.com/ru/explore?page=2.(date of access: 17.03.2023)

ДОДАТКИ

ДОДАТОК А

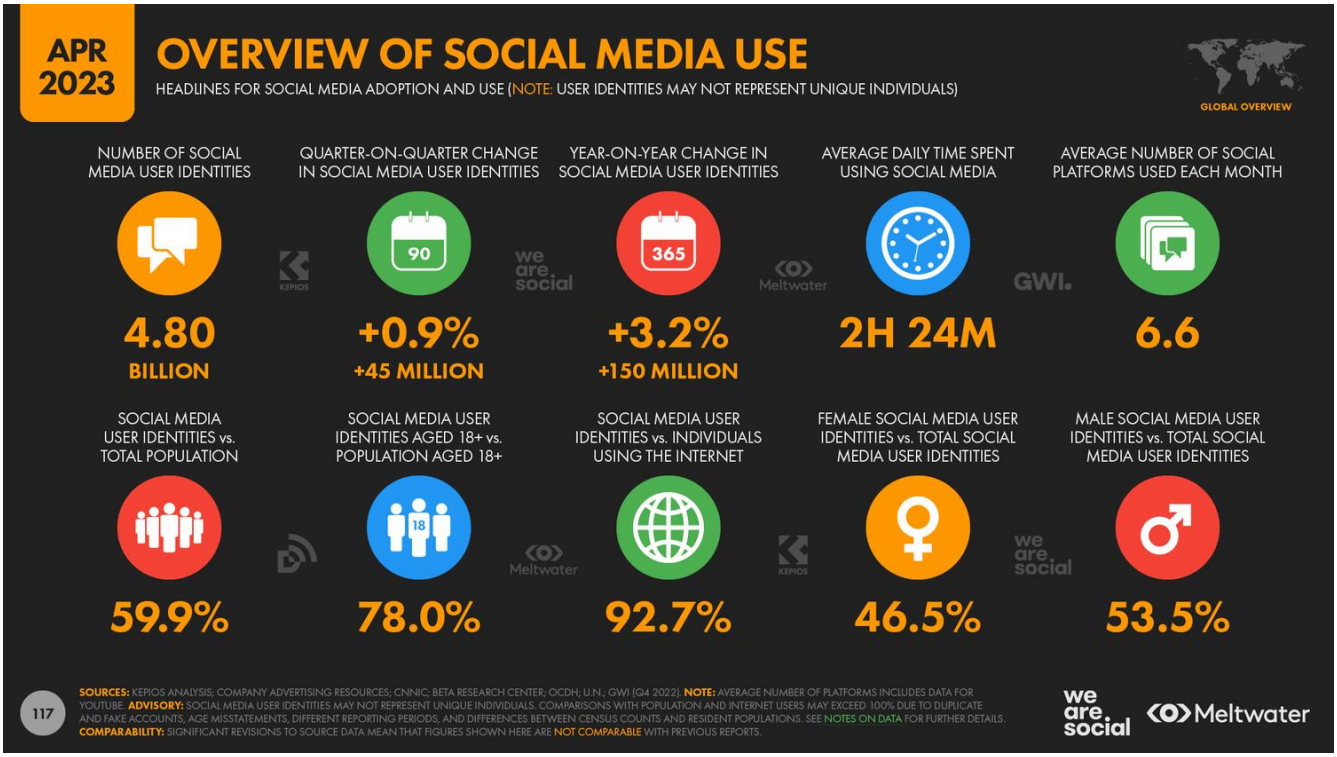


Рисунок А1 – Статистика використання соціальних мереж у світі станом на 2023 рік

ДОДАТОК Б

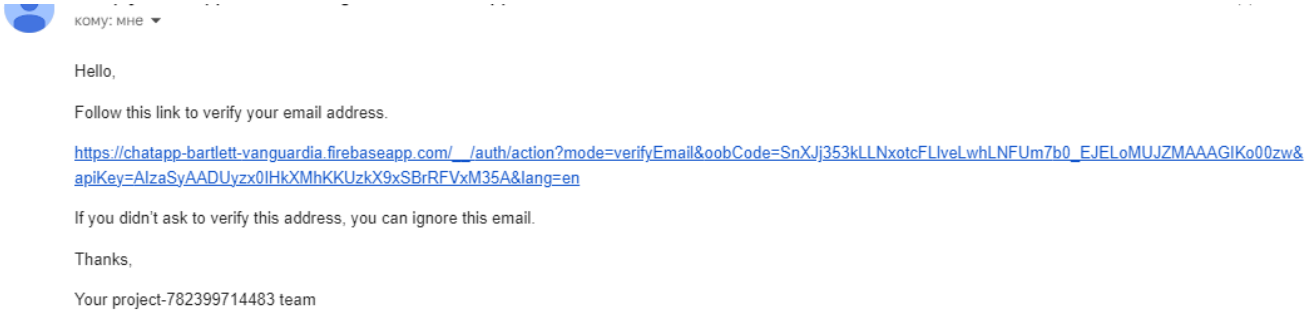


Рисунок Б1 – Приклад повідомлення при реєстрації від Firebase

ДОДАТОК В

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<ContentPage
    x:Class="ChatApp.Pages.Auth.Signup"
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:ChatApp">

    <StackLayout BackgroundColor="#ffffff">

        <StackLayout
            BackgroundColor="Transparent"
            HorizontalOptions="Start"
            VerticalOptions="Start">
            <StackLayout
                Grid.Row="0"
                Margin="10,20,10,0"
                Orientation="Horizontal">
                <StackLayout
                    HorizontalOptions="StartAndExpand"
                    Orientation="Horizontal"
                    VerticalOptions="CenterAndExpand">

                    <Image
                        BackgroundColor="transparent"
                        HeightRequest="30"
                        HorizontalOptions="StartAndExpand"
                        Source="back.png"
                        VerticalOptions="CenterAndExpand" />
                    <Label
                        FontAttributes="Bold"
                        FontSize="Small"
                        HorizontalOptions="EndAndExpand"
                        Text="Back"
                        TextColor="#424242"
                        VerticalOptions="CenterAndExpand" />

                    <StackLayout.GestureRecognizers>
                        <TapGestureRecognizer Tapped="Btn_Back" />
                    </StackLayout.GestureRecognizers>
                </StackLayout>

            </StackLayout>
        </StackLayout>
```

Рисунок В1 – фрагмент розмітки сторінки аутентифікації застосунку

```
class ValidateEmail
{
    Ссылок: 3
    public static bool IsValidEmail(string email)
    {
        if (string.IsNullOrWhiteSpace(email))
            return false;

        try
        {
            // Normalize the domain
            email = Regex.Replace(email, @"(@)(.+)$", DomainMapper,
                                  RegexOptions.None, TimeSpan.FromMilliseconds(200));

            // Examines the domain part of the email and normalizes it.
            string DomainMapper(Match match)
            {
                // Use IdnMapping class to convert Unicode domain names.
                var idn = new IdnMapping();

                // Pull out and process domain name (throws ArgumentException on invalid)
                string domainName = idn.GetAscii(match.Groups[2].Value);

                return match.Groups[1].Value + domainName;
            }
        }
        catch (RegexMatchTimeoutException e)
        {
            Console.WriteLine(e);
            return false;
        }
        catch (ArgumentException e)
        {
            Console.WriteLine(e);
            return false;
        }
```

Рисунок В2 – фрагмент коду файлу, який відповідає за перевірку електронної пошти