НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізу

Кафедра системного проектування

До захисту допущено:

Завідувач кафедри

_____ А.І. Петренко

«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою

«Інтелектуальні сервіс-орієнтовані розподілені обчислювання»

спеціальності 122 «Комп'ютерні науки»

на тему: «Кросплатформений мобільний додаток з налаштуванням користувачем функціональних можливостей для спілкування »

Виконала: студентка IV курсу, групи ІТ-ЗП81
Гаврилюк Катаріна Вікторівна _____
(підпис)

Керівник:доцент, кандидат технічних наук,
Капшук Олег Олексійович _____
(підпис)

Консультант економічний: доцент, к.е.н.,
Рощина Надія Василівна _____
(підпис)

Рецензент: професор, д.т.н.,
Бідюк Петро Іванович _____
(підпис)

засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра системного проектування

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Інтелектуальні сервіс-орієнтовані розподілені обчислювання»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ А.І. Петренко

«___»_____2020 р.

ЗАВДАННЯ

на дипломну роботу студенту

Гаврилюк Катаріна Вікторівна

1. Тема роботи «Кросплатформений мобільний додаток з налаштуванням користувачем функціональних можливостей для спілкування», керівник роботи Капшук Олег Олексійович, кандидат технічних наук, доцент, затверджені наказом по університету від «01» липня 2020р. № 1623-ф.

2. Термін подання студентом роботи 7 грудня 2020 року

3. Вихідні дані до роботи

Дослідження інтерфейсу чатів у мобільних месенджерах з метою створення нової деревоподібної структури чатів.

4. Зміст роботи

Аналіз популярних у Україні мобільних месенджерів. Аналіз технологій для побудови кросплатформеного мобільного додатку. Розробка архітектурного рішення для додатку. Реалізації прототипу кросплатформенного мобільного месенджера з налаштуванням кількості вкладених чатів.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) Презентація до захисту роботи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., к.е.н., доцент		

7. Дата видачі завдання 01.08.2020

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	01.08.2020	
2	Аналіз предметної області	10.08.2020	
3	Збір інформації	15.08.2020	
4	Аналіз існуючих методів	25.10.2020	
5	Власне дослідження засобів	01.11.2020	
6	Функціонально-вартісний аналіз	15.11.2020	
7	Оформлення дипломної роботи	1.12.2020	

Студент _____ Гаврилюк К.В.
(підпис) (ініціали, прізвище)

Керівник роботи _____ Капшук О.О.
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

До бакалаврської дипломної роботи Гаврилюк Катаріни Вікторівни

на тему:

"Кросплатформений мобільний додаток з налаштуванням користувачем функціональних можливостей для спілкування"

Спілкування є великою частиною життя кожної людини і тенденції для збільшення онлайн спілкування лише поглиблюється завдяки карантину та глобальній ситуації в цілому. Три найпопулярніших в Україні мобільних месенджера незважаючи на широкий функціонал не дають користувачеві змінювати саму структуру чатів.

Метою даної дипломної роботи є створення кросплатформенного програмного продукту з налаштуванням користувачем функціональних можливостей для спілкування.

У роботі проведено аналіз платформ для створення клієнтської та серверної частин застосунку, описано технології та принципи архітектурної побудови програмного продукту.

Розроблено прототип кросплатформенного мобільного месенджеру з інноваційною гілкоподібною структурою підчатів.

Загальний об’єм роботи 105 с., 2 д., 40 рис., 6 табл., 26 джерел.

Ключові слова: мобільний месенджер, кросплатформа, чат, Unity, GCP, PostgreSQL.

АННОТАЦИЯ

К бакалаврской дипломной работе Гаврилюк Катарины Викторовны

на тему:

"Кроссплатформенное мобильное приложение с настройкой пользователем функциональных возможностей для общения"

Общение – это большая часть жизни каждого человека, и тенденции для увеличения онлайн общения только усугубляется благодаря карантину и глобальной ситуации в целом. Три самых популярных в Украине мобильных мессенджера, несмотря на широкий функционал, не дают пользователю изменять саму структура чатов.

Целью данной работы является создание кроссплатформенный программного продукта с настройкой пользователем функциональных возможностей для общения.

В работе проведен анализ платформ для создания клиентской и серверной частей приложения, описаны технологии и принципы архитектурного построения программного продукта.

Разработан прототип кроссплатформенного мобильного мессенджера с инновационной древовидной структурой подчатов.

Общий объем работы 105 с., 2 д., 40 рис., 6 табл., 26 источников.

Ключевые слова: мобильный мессенджер, кроссплатформа, чат, Unity, GCP, PostgreSQL.

ANNOTATION

On Gavrilyuk Katarina Victorivna bachelor's degree thesis:
"Cross-platform mobile application with customization of communication’s functionality by user "

Communication is a big part of everyone's life, and the trend to increase online communication is only exacerbated by quarantine and the global situation as a whole. The three most popular mobile messengers in Ukraine, despite the wide functionality, do not allow the user to change the very structure of chats.

The purpose of this work is to create a cross-platform software product with customizable user functionality for communication.

This paper analyzes the platforms for creating client and server parts of the application, describes the technologies and principles of the software product's architecture.

A prototype cross-platform mobile messenger with an innovative tree-like structure of subchats has been developed.

The total volume of work 105 pages, 2 applications, 40 figures, 6 tables, 26 sources.

Keywords: mobile messenger, cross-platform, chat, Unity, GCP, PostgreSQL.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ММ – мобільний месенджер

SMS (short message service) – послуга коротких повідомлень

API (application programming interface) – інтерфейс прикладного програмування

AWS (Amazon Web Services) – веб-служби Amazon

GCP (Google Cloud Platform) – Хмарна платформа Google

MVC (Model-View-Controller) – Модель–представлення–контролер

SQL (Structured Query Language) – Мова структурованих запитів

NoSQL (Not only SQL) – не тільки SQL

ВСТУП

Спілкування є основою будь-якої взаємодії і невід’ємною частиною людського життя. Сьогодні важко уявити собі людину у якої немає мобільного смартфону зі встановленими мобільними додатками. Можливості мобільних додатків дозволяють інтернет-користувачам отримувати і передавати великий обсяг інформації в режимі реального часу. Тенденції, що формуються в рамках широкого поширення інформаційних технологій, говорять про актуальні сьогодні мобільні додатки – месенджери.

Месенджер – програма, мобільний додаток або веб-сервіс з можливістю здійснення дзвінків, миттєвого обміну текстовими та голосовими повідомленнями, відправленням зображень та відео, геоданими тощо.

Актуальність теми

За даними дослідженнь, середній час, який користувач проводить в месенджерах за 2019 рік, склав 48 хвилин на добу, не враховуючи соціальні мережі[1]. Цей показник буде вищій у 2020 році за рахунок пандемії та ще більшого перенесення спілкування у онлайн. Зокрема внаслідок карантинів і об'єктивного переходу робочого процесу в онлайн навіть в тих галузях де раніше не було такої потреби. Це зумовлює підвищення інформаційного навантаження на людину і потребує розвитку можливостей по структуруванню інформації, зокрема у чатах. Створення чатів у чаті дозволять структурувати спілкування користувачів та спростять пошук по діалогу. Нині популярні месенджери не оснащені подібною функцією, тому таке нововведення спростить спілкування між користувачами.

Мета

Мета даної дипломної роботи створити і описати прототип кросплатформеного мобільного месенджера з функціоналом створення "підчатів" (чат в чаті) з основного чату, з можливістю легкого доступу до підчату у користувача.

Основні завдання

- Дослідити функціональні можливості розповсюджених мобільних месенджерів
- Проаналізувати технічні рішення для створення кросплатформенного застосунку
- Розробити архітектуру кросплатформеного мобільного месенджера
- Створити адаптований інтерфейс
- Реалізувати прототип

1 Аналіз сучасних мобільних додатків для спілкування

1.1 Шляхи та засоби спілкування користувачів із застосування мобільних пристроїв

Невід'ємною частиною повсякденного життя кожної людини є спілкування. Спілкування (від латинського communicare, що означає «ділитися») - це передача будь-якої форми інформації від однієї особини чи групи до іншої особини чи групи за допомогою знаків, символів, звуків чи зрозумілих для обох сторін правил.

Люди спілкувалися с давніх часів причому декількома способами одночасно:

за допомогою зображень - наскальний живопис, картини, фото, смайли, GIF, емоджі.

за допомогою голосу - живе спілкування віч-на-віч, телефонна розмова, автовідповідачі, голосові повідомлення.

за допомогою тексту - листи, автобіографічні твори, телеграми, смс, текстові повідомлення.

До появи першого телефонного апарату не було можливо передавати і отримувати голосові повідомлення, у 1860 році це змінив Антоніо Меуччі. Останнього вважають винахідником першого приладу “Teletrofono”, що був здатний передавати звуки по електричних проводах. Випадкове відкриття при лікуванні головного болю у одного зі своїх пацієнтів електрошоком Меуччі використав для створення апаратів для спілкуватися зі своєю хворою дружиною знаходячись у різних кімнатах [2].

Довгий час винахідником телефонного апарату вважався Олександр Белл. Його винахід, на 16 років пізніше за Меуччі, міг передавати голос на відстань до 500 метрів. Незважаючи на можливість з 1881 року використовувати телеграфні мережі для телефонних розмов телефони перша трансатлантична телефона розмові відбулась між Лондоном та США у 1927 році [3].

У 1946 році в Америці з'явився прототип першого мобільного телефону, з якого можна було телефонувати з автомобіля. Мобільний телефон використовує для відправлення та отримання дзвінків радіочастотну лінію. І перший портативний мобільний телефон був продемонстрований Джоном Ф. Мітчеллом [4] та Мартіном Купером з Motorola в 1973 році, використовуючи телефон вагою 2 кілограми.

З розвитком технологій змінювалась і структура спілкування. Якщо на початку становлення ери телефонного зв’язку телефон використовувався виключно для розмови між двома абонентами, то з розвитком нових технологій спілкування таких як SMS та MMS користувачі почали використовувати і обмін повідомленнями. SMS та MMS дозволяли користувачам мобільних телефонів надсилати текстові чи медійні повідомлення без підключення до інтернет але вони тарифікувалися згідно з тарифами операторів мобільного зв’язку.

Перші мобільні телефони могли приймати лише дзвінки та повідомлення, а сучасні телефони - смартфони це поєднання персонального комп'ютера та телефона. Вони можуть приймати дзвінки, отримувати пошту, проводити оплату послуг та є платформою для встановлення різноманітних мобільних додатків. У сучасному світі стрімкого технологічного прогресу велика частина спілкування відбувається за допомогою смартфонів.

Мобільний додаток - це програма що може бути встановлена на будь-який мобільний пристрій, як то смартфон, планшет, тощо [5].

Розвиток мобільного інтернету та смартфонів зробив великий поштовх для створення і застосування мобільних месенджерів.

Мобільний месенджер ММ - це мобільний додаток, що дозволяє обмінюватися текстовими, графічними та аудіальними повідомленнями. На відміну від SMS та MMS користувач не платить за кожне повідомлення, а тарифікується лише інтернет трафік, що використовується при відправленні, якщо користувач використовує мобільний інтернет. При під’єднані телефона до бездротової мережі інтернету Wi-Fi відправлення повідомлень є умовно безкоштовним (мається на увазі, що хтось все

одно сплачує вартість доступу до мережі інтернет). У 2012 році обсяг SMS досягши піку, почав поступово зменшуватися аж поки в 2013 році ММ перевершили SMS за загальним обсягом повідомлень.

Аналіз компанії MetrixLab (рис 1.1) показує що 63% Baby Boomers (1946 to 1964 роки народження) використовують ММ як заміну платним текстовим повідомленням. Це є вагомим показником бо це є люди старшого віку. Двоє з трьох народжених з 1981 по 2000 роки також використовують ММ замість SMS.

ММ також замінюють платні телефонні дзвінки - 67% людей народжених з 2000 і 66% людей що народилися з 1981 по 2000 використовують програми обміну повідомленнями як заміну дзвінкам [6].

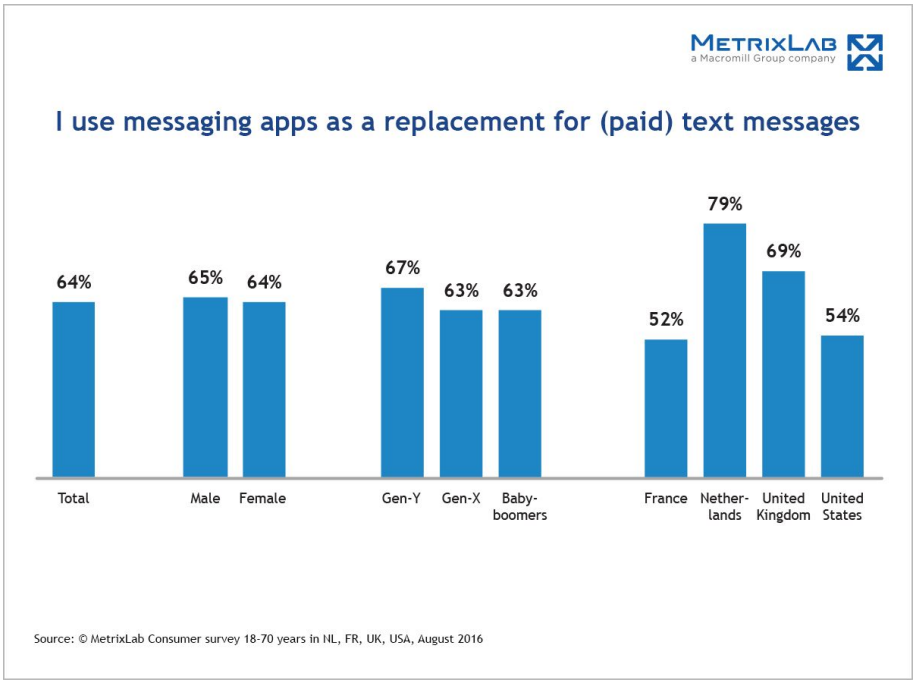


Рисунок 1.1 – Розподіл користування ММ серед респондентів, які народилися з 1946 по 2000 рік згідно даних аналітичної компанії MetrixLab

З того часу програми обміну повідомленнями розширили можливості для користувачів. Окрім більш зручної альтернативи текстовим повідомленням на основі SMS, він також пропонує розширені функції, такі як групові чати, голосові

дзвінки, відеодзвінки та обмін фотографіями. Статистика за серпень 2020 року показує, що понад 41 мільйон мобільних повідомлень надсилається за хвилину онлайн для найпопулярнішого у світі месенджера WhatsApp, що має 1,6 млрд активних користувачів на місяць [7].

На сьогодні функціонал ММ розширився і деякі ММ перетворилися на широкі платформи, що дозволяють оновлювати статус, залучати до розмови чат-ботів, проводити платежі.

Зараз програми обміну повідомленнями мають більше глобальних користувачів, ніж традиційні соціальні мережі, а це означає, що вони будуть відігравати все більшу роль у спілкуванні в майбутньому.

1.2 Огляд мобільних додатків для спілкування

Розглянемо найбільш популярні ММ, такі як WhatsApp, Telegram, Viber, Line. WeChat та QQ Messenger розповсюджені у Китаї, що робить їх ММ з великою кількістю користувачів, але вони є локальними і не актуальні для Європи чи Америки, тому розглядати ми їх не будемо. Майже всі сучасні ММ мають версію для персонального комп’ютера (desktop) яка була або попередником ММ або з’явилась пізніше, як додаткова послуга.

Деякі програми акцентують увагу на певних функціях- наприклад, Skype та Zoom позиціонують себе як додатки для відеодзвінків, Slack, Microsoft teams та Facebook Workplace - на обміні повідомленнями та спільному використанні файлів для робочих груп, а Snapchat - на графічних повідомленнях, які зникають через певний проміжок часу. Всі соціальні мережі пропонують послуги обміну повідомленнями, наприклад Facebook має окремий додаток Facebook Messenger, тоді як інші мають функцію прямого обміну повідомленнями як додатковий допоміжний компонент своїх платформ соціальних мереж, наприклад Instagram, Reddit, TikTok і Twitter, тобто реалізація здійснюється або безпосередньо або через чати.

1.3 Огляд існуючих програмних рішень

1.3.1 WhatsApp Messenger

Це американська безкоштовна програма, заснована у 2009 році американським програмістом українського походження Яном Кумом та Браяном Ектоном, куплена у 2014 році компанією Facebook, Inc, що дозволяє користувачам надсилати текстові та голосові повідомлення, здійснювати голосові та відеодзвінки, а також обмінюватися зображеннями, документами, місцезнаходженням користувачів та іншими засобами масової інформації. [8]

Користувачі можуть створювати групи як то Родина, Друзі чи, наприклад, Вихідні в Карпатах, розміром до 256 учасників та модифікувати сповіщення. Вбудована в програму камера дозволяє робити знімки та відправляти їх одразу у додатку. Максимальний розмір для відправлення файлу складає 100 Мб. Для деяких країн дозволені грошові транзакції в середині ММ. Перелік основних функціональних можливостей зображений на рисунку 1.2.

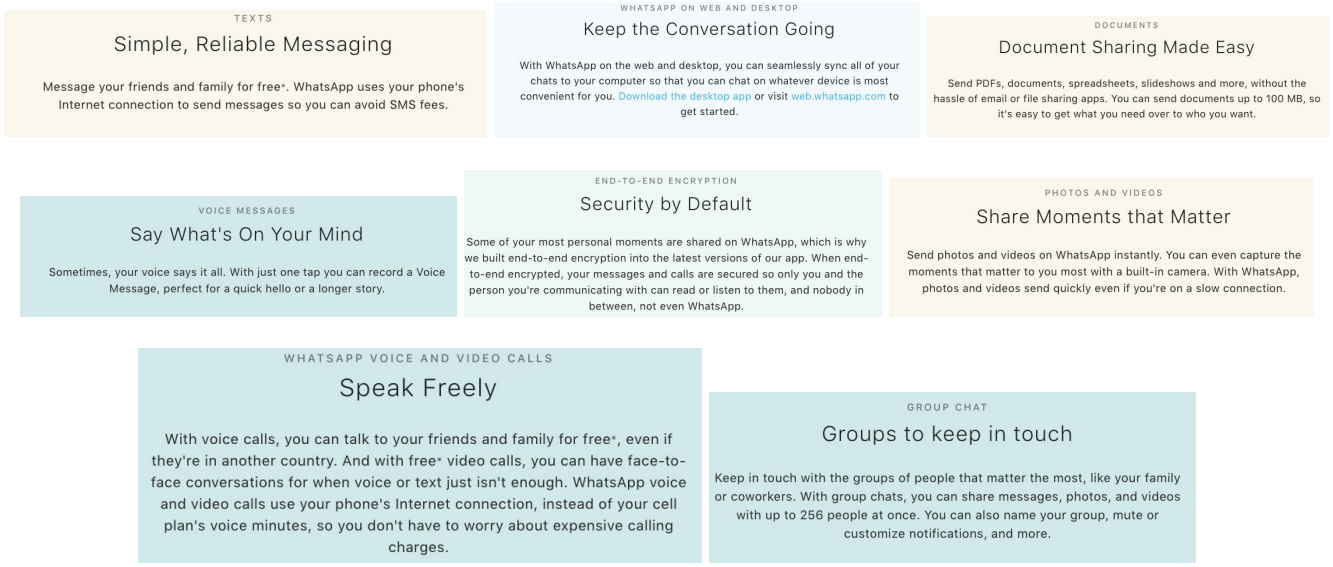


Рисунок 1.2 – Функціональні можливості WhatsApp

Клієнтська програма WhatsApp працює на мобільних пристроях, але також доступна з настільних комп’ютерів, доки мобільний пристрій користувача

залишається під’єднаним до Інтернету. ММ вимагає від користувачів надання стандартного номера мобільного зв’язку для реєстрації в службі. У січні 2018 року WhatsApp випустив самостійний бізнес-додаток, орієнтований на власників малого бізнесу, під назвою WhatsApp Business, щоб дозволити компаніям спілкуватися з клієнтами, які використовують стандартний клієнт WhatsApp [8].

Перший рік користування є безкоштовним, надалі є абонентська плата в розмірі 1 доллар на рік і не містить реклами.

ММ має достатньо інтуїтивний інтерфейс. При реєстрації треба ввести номер телефона, отримати СМС с кодом підтвердження, це характерно для усіх ММ. При натисканні на контакт відкривається віконце чату з ним. Поруч з аватаркою яка автоматично підтягується з телефонної книги зазначено час, коли контакт останній раз був в мережі. Також на 3 скріншоті видно, що є функції налаштування нотифікування та автоматичного знищення повідомлень. У нижній частині екрану знаходяться кнопки вкладення та моментальне фото з правої сторони та смайли з лівої сторони.

Можна вставити в чат картинку, відео, записаний голос, картку контакту і своє місце розташування, та налаштувати свій статус. Приклад інтерфейсу зображено на рисунку 1.3.

WhatsApp використовує наскрізне шифрування за допомогою протоколу Signal, що розроблявся компанією Open Whisper Systems для шифрування голосових викликів, відеозвязку та відправки повідомлень. Доречі дочірня компанія Signal створила свій власний ММ, що вважається найзахищенішим ММ у всіті [9].

Для шифрування протокол використовує алгоритм подвійного храповика, попередники ключів, розширений протокол потрійного обміну ключами Діффі-Хеллмана та AES-256[9].

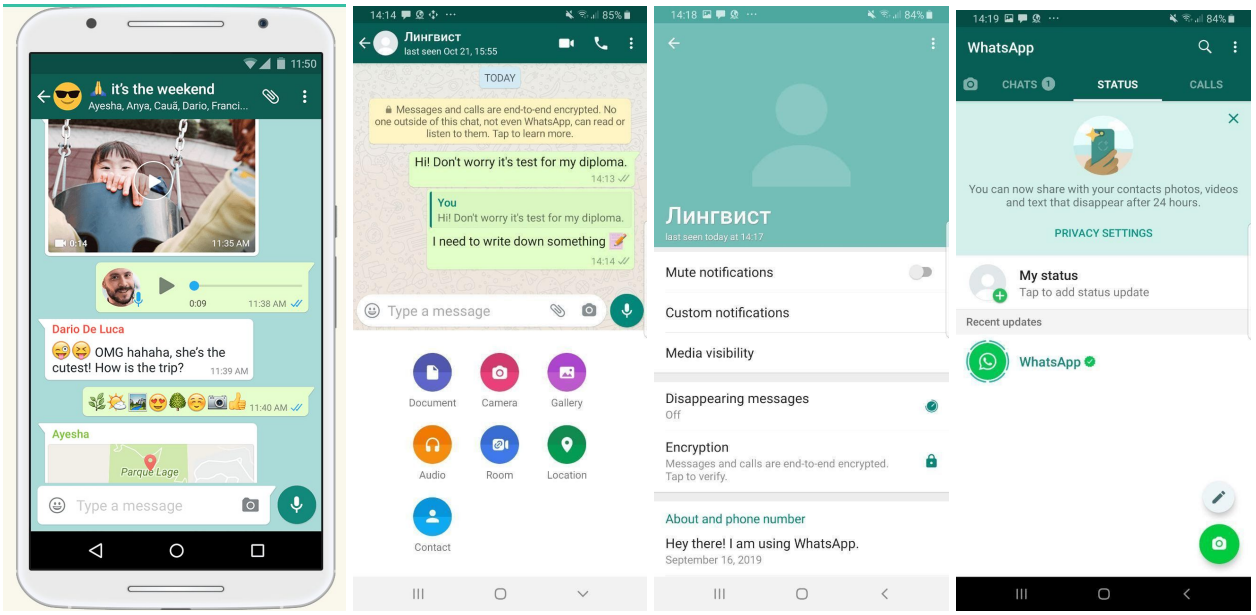


Рисунок 1.3 – Приклад інтерфейсу Whatsapp

За допомогою Алгоритму “Подвійний храповик” реалізується пряма та зворотня секретність. Назва алгоритму є відсиланням до механічної шифрувальної машини Enigma, в якій використовувалися храповики - шестерні з похилими зубцями, що рухаються тільки в одному напрямку. За рахунок цього в шифрувальній машині уникався стан шестерень, що повторював один з недавно використаних.

У ММ часто змінюються сесійні ключі, кожен з яких шифрує свою невелику порцію повідомлень, для того щоб при компрометуванні ключа шифрування поточного повідомлення можна було розшифрувати лише невелику кількість повідомлень - це пряма секретність. В той час, як зворотна секретність забезпечує захист майбутніх повідомлень при компрометуванні поточного ключа. Нові ключі генеруються таким чином, що їх взаємозв'язок з попередніми обчислити вкрай складно. Алгоритм часто змінює сесійні ключі, при цьому не даючи повторно використовувати раніше згенеровані[10].

Враховуючи, що WhatsApp, дозволяє здійснювати перекази між рахунками у фінансових установах, номери карток і банків зберігаються в зашифрованому вигляді. Однак, оскільки фінансові установи не можуть обробляти операції, не

отримуючи інформацію, пов’язану з цими платежами, ці платежі не шифруються наскрізно.

WhatsApp не зберігає повідомлення на своїх серверах та вони знищуються одразу після доставки.

1.3.2 Telegram Messenger

На сайті виробника пишуть, що “Telegram - це програма обміну повідомленнями, орієнтована на швидкість та безпеку, надзвичайно швидка, проста та безкоштовна. Ви можете використовувати Telegram на всіх своїх пристроях одночасно - ваші повідомлення синхронізуються на будь-якій кількості ваших телефонів, планшетів або комп’ютерів.” [11]

Користувач може надсилати повідомлення, фотографії, відео та файли будь-якого типу (doc, zip, mp3 тощо) до 2 Гб, а також створювати групи до 200 000 людей або канали для трансляції необмеженій аудиторії (рис 1.4).

На відміну від WhatsApp, Telegram - це хмарний месенджер з безперебійною синхронізацією, тому всі повідомлення можна переглядати одночасно на всіх пристроях.

Для того щоб знайти людину в Telegram треба знати або номер його телефона або ім’я користувача. ММ позиціонує себе як поєднання SMS та електронної пошти.

Має відкритий API і окремо платформу для розробників API Bot, що допомагає у створенні спеціалізованих інструментів для Telegram, інтегруванні будь-яких сервісів та приймання платежів від користувачів по всьому світу.

Є абсолютно безкоштовним та не містить реклами.

Рисунок 1.4 – Функціональні можливості Telegram Messenger

Після подібної до Whatsapp реєстрації користувач може додатково придумати собі ім’я користувача за яким інші користувачі програми можуть знаходити його в мережі. В налаштуваннях можлива зміна як номера телефона так і імені чи фото. Також можна змінити колір оформлення UI.

У вкладці "Конфіденційність та безпека" є функція Самознищення аккаунта. Якщо користувач не використовував аккаунт протягом обраного періоду (1 місяць, 3 місяці, півроку, рік), то аккаунт з усіма даними і листуванням повністю видаляється. Не обов’язково зберігати всі медіа файли на телефоні, в налаштуванні кешу можна виставити дату, після закінчення терміну якої, кеш автоматично очиститься. Якщо це не секретний чат, то всі файли залишаються у хмарі, тому їх можна повторно завантажити.

Кожен користувач може створити свій публічний канал. Останній це не чат і не група, інші користувачі можуть лише читати канал та зберігати звідки повідомлення, коментування безпосередньо у каналі відсутне, але власник може відкрити комендаті безпосередгьо до кожного повідомлення. У самому каналі можна налаштувати щось на кшталт лайків.

Приклад інтерфейсу Telegram: створення публічного каналу, загальний вигляд і функціонал чатів та приклад функціонування боту зображено на рисунку 1.5 [12].

Для шифрування Telegram використовує MTProto, що підтримує два рівні: шифрування клієнт-сервер, яке використовується в хмарних чатах Telegram, та наскрізне шифрування, яке використовується в таємних чатах Telegram. MTProto - це криптографічний протокол, який використовується в системі обміну повідомленнями Telegram для шифрування листування користувачів. В основі протоколу лежить оригінальна комбінація симетричного алгоритму шифрування AES (в режимі IGE), протокол Діффі-Хеллмана для обміну 2048-бітними RSA-ключами між двома пристроями та ряд хеш-функцій. Протокол допускає використання шифрування end-to-end з опціональною звіркою ключів [11]. На відміну від WhatsApp де наскрізне шифрування відбувається у кожному чаті Telegram шифрує так лише таємні чати. Також питання зберігання інформації з хмарних чатів залишається відкритим.

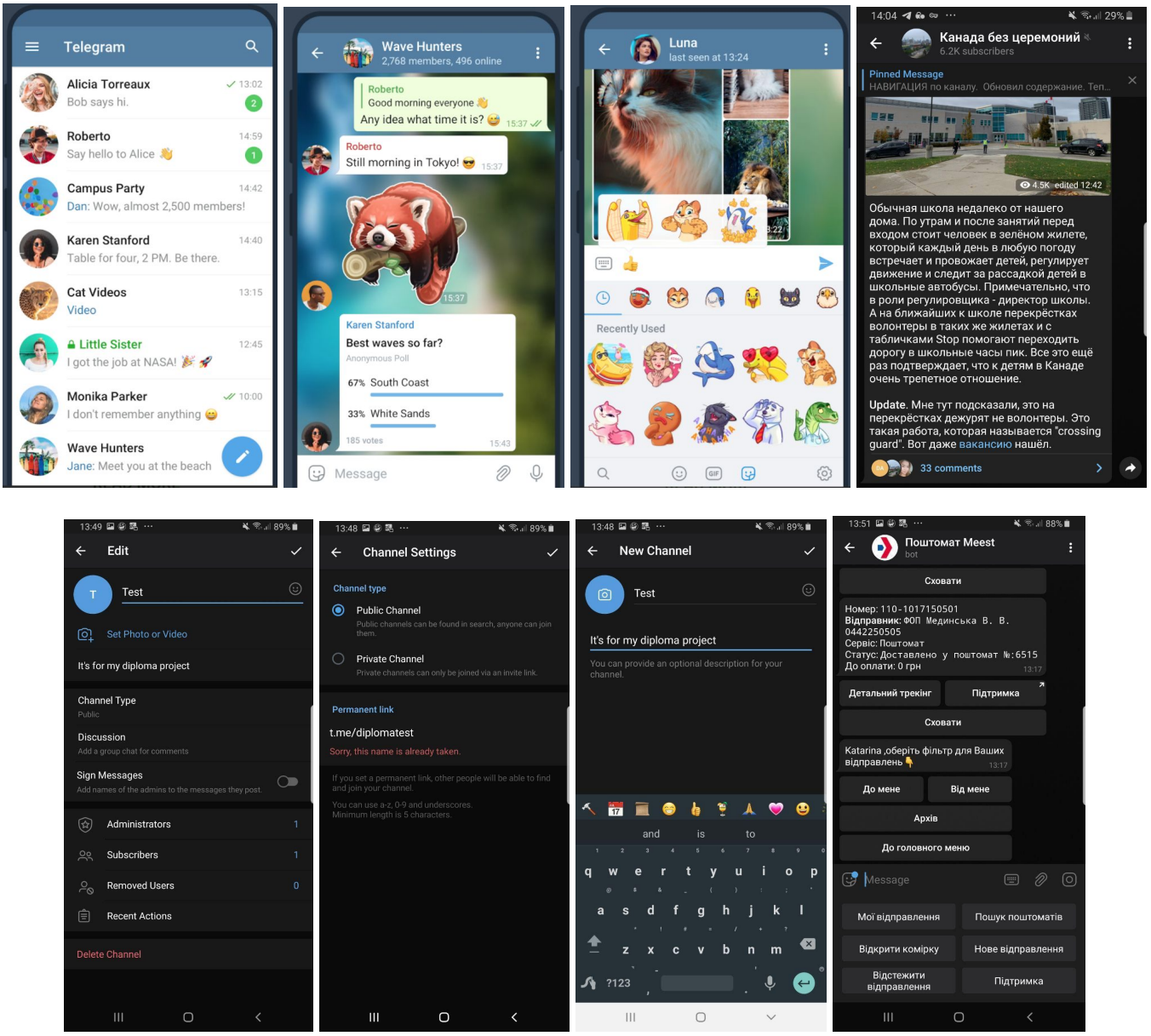


Рисунок 1.5 – Приклад інтерфейсу Telegram Messenger

1.3.3 Rakuten Viber Messenger

З 2017 року Viber належить японській компанії Rakuten. Як і решта ММ дозволяє обмінюватися миттєвими повідомленнями, робити дзвінки всередині мережі, створювати чати. Користувачі реєструються та ідентифікуються за номером телефону, хоча послуга доступна на десктоп платформах без необхідності мобільного зв'язку. На відміну від всіх попередніх ММ забезпечує платну послугу

міжнародних стаціонарних та мобільних дзвінків під назвою Viber Out за допомогою якої користувач може зателефонувати на будь-який телефон з додатку [13]. Viber є найпопулярнішим ММ в Україні [14]. Також додаток дозволяє проводити платежі. Частина функціоналу зображена на рисунку 1.6 [13].

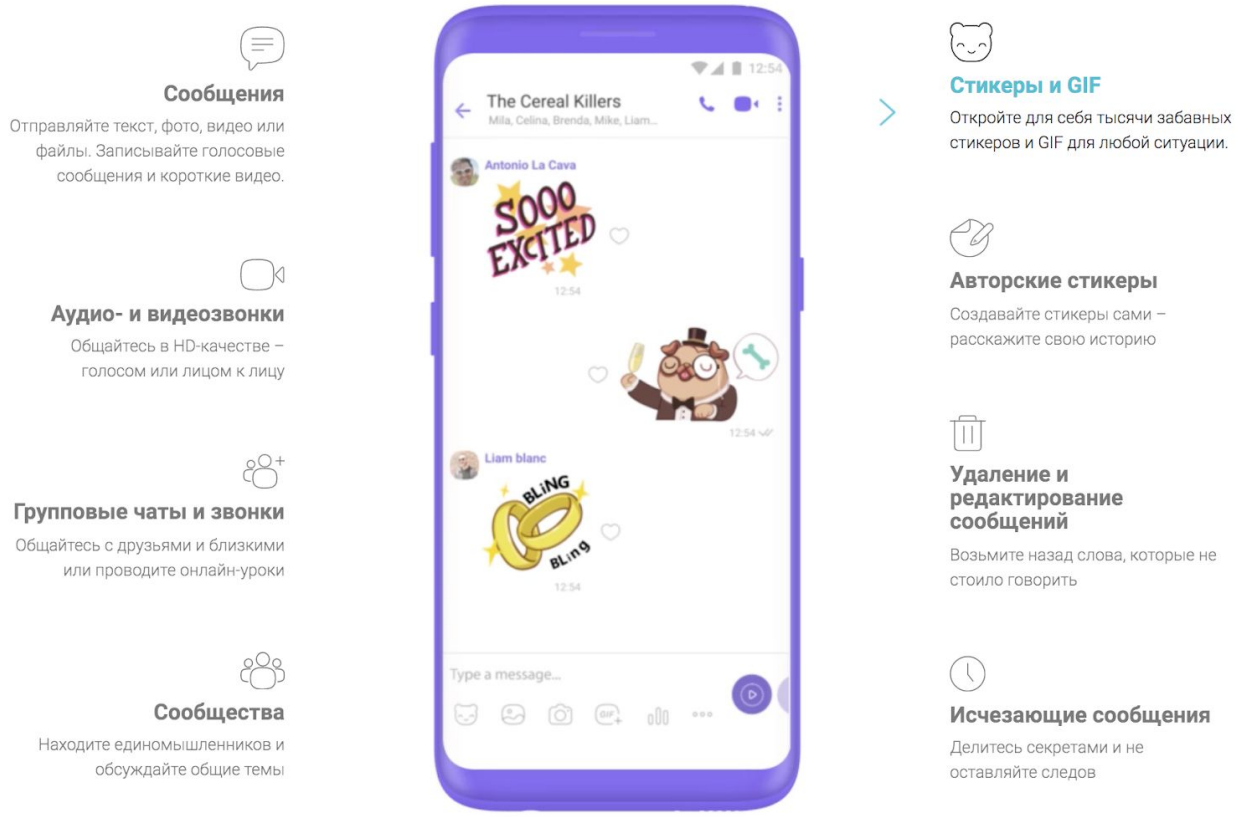


Рисунок 1.6 – Функціональні можливості Rakuten Viber Messenger

Окремо треба сказати про Viber чати (рис 1.7) , які є дуже популярними в Україні. Користувачі можуть налаштовувати права для учасників чату, видаляти повідомлення, ставати на назначати модераторів. Окремо можна виділити перегляд статистики товариств створення повідомлень що будуть зникати після певного проміжку часу, а також сповіщення про скріншот екрану.

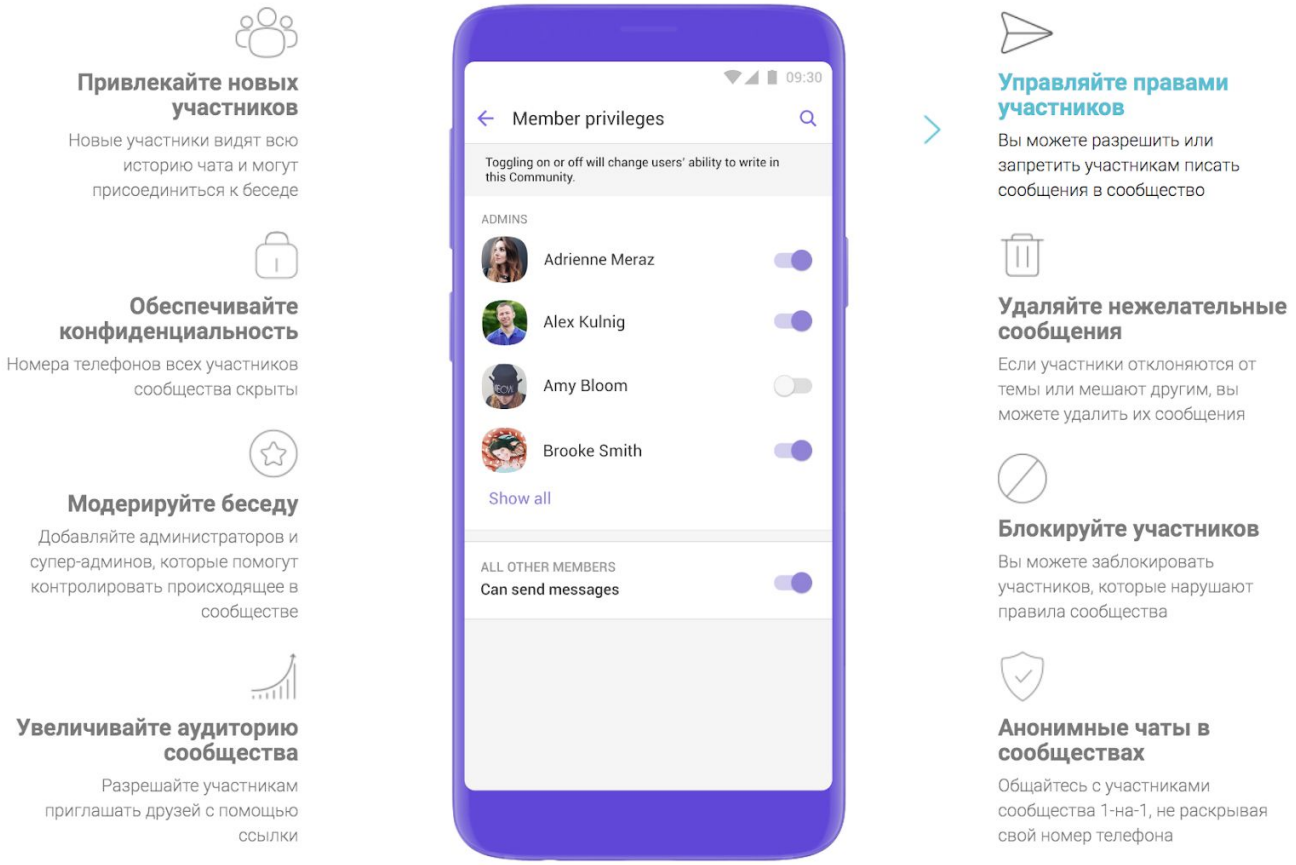


Рисунок 1.7 – Функціонал Rakuten Viber Messenger чату

Сервіс є безкоштовним, але продає стікери та містить рекламу. За рахунок того, що щоб добавити людину в чат треба просто знати її номер телефону містить дуже багато спаму.

Реєстрація в Rakuten Viber відбувається так само як і у попередніх ММ за номером телефону.

Навігаційне меню знаходиться знизу: чати; виклики; розділ новин, стікерів, корисної інформації та реклами; кнопка More після натискання на яку користувач знаходить налаштування та іншу інформацію про себе. ММ також повідомляє коли хтось з телефонної книги має дні народження, що по функціоналу схоже на соціальні мережі. Кнопка Viber Out дозволяє робити виклики на стаціонарні та мобільні телефони на яких немає ММ Viber.

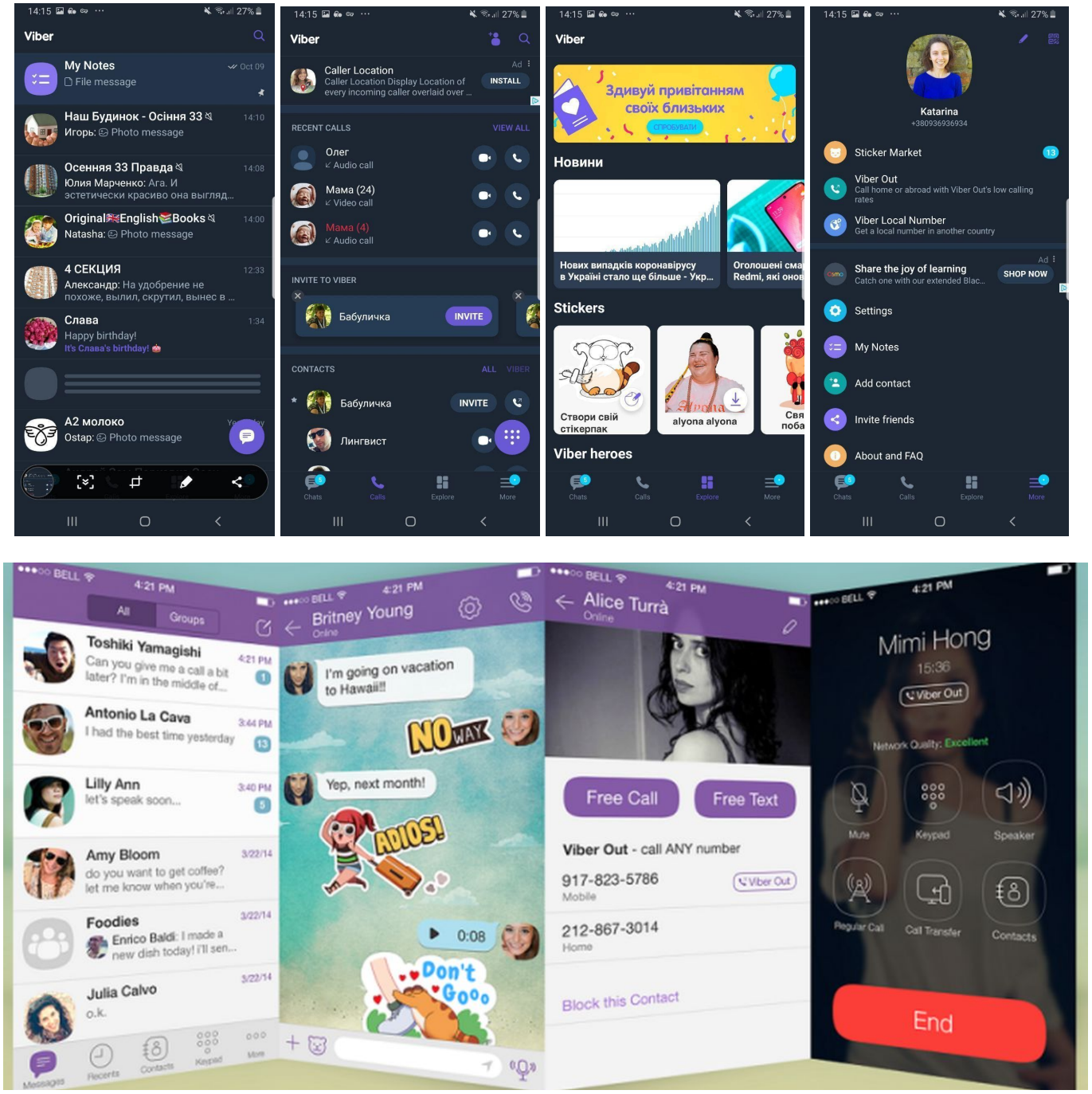


Рисунок 1.8 – Приклад інтерфейсу Rakuten Viber Messenger

Rakuten Viber використовує наскрізне шифрування за замовчуванням для усіх чатів, а у секретних чатах є функції автоматичного видалення повідомлень, заборони або відстеження скріншотів, захист від копіювання та пересилань повідомлень тощо. Для створення резервних копій історії переписки Viber пропонує використовувати Google Drive.

Viber використовує протокол Proteus - по суті, той же Signal з іншими криптографічними примітивами [13][9].

1.4 Висновки до розділу

В даному розділі було зроблено огляд функціональних можливостей найпопулярніших у світі, та зокрема Україні, ММ. Всі вони пропонують широкий спектр можливостей для користувача - відправка та отримання повідомлень із будь-який типом інформації починаючи від звичайного тексту закінчуючи геоміткою місцезнаходження; дзвінки як у середині мережі так і на звичайні мобільні чи стаціонарні телефони по всьому світу; створення каналів та чатів як альтернатива сучасним медіа та соц мережам; різноманітні боти по пошуку будь-якого контенту, відстежень відправлень та замовлень; навіть можливість оплати. Проте в жодного з них немає можливості створення підчату для обговорення конкретної теми чи питання, що б значно полегшило спілкування зменшивши кількість повідомлень у загальному чаті.

2 Вибір засобів для розробки прототипу мобільного месенджера

2.1 Побудова архітектури застосунку для реалізації загальної ідеї мобільного месенджера

Для побудови ММ ми використаємо архітектуру стандартного мобільного додатку, яку можно умовно розділити на три частини, що зображені на рисунку 2.1 : сам додаток мобільного клієнту, серверна частина та база даних для зберігання повідомлень та іншої інформації

Рисунок 2.1 – Архітектура мобільного додатку

Ми будемо створювати крос-платформний ММя, тому треба підібрати такий набір інструментів (фреймворк), який дозволить створити додаток, що підходить відразу для iOS, Android та вебу, враховуючи можливості і обмеження, що є у кожного фреймворка.

Головна ідея проекту заключається у створенні гілкоподібної структури чатів де з основного чату може бути створено підчати 1 рівня, а за потреби з підчату 1 рівня можна будет створити підчат 2, 3 чи 4 рівня (Рис. 2.2). Це дасть змогу користувачам одночасно спілкуватися на різні теми окремо, не змішуючи всі повідомлення у один великий чат, і не створювати нові чати. Якщо створювати

додаткові чати то їх кількість з часом буде дуже велика, наприклад, наша інститутська група вела спілкування у загальному чаті де ми обговорювали навчання, в тому числі лабораторні роботи, деякі предмети мали по 5 лабораторних робіт, на додачу курсові та розрахунково-графічні роботи. Якщо підрахувати кількість чатів, які ми мали б створити для одного семестру, то число сягне більше 40, не говорячи про те, що у кожного з нас є особисті і робочі чати та чати для спілкування з друзями.

Також, деякі питання при обговоренні потребують часу для пошуку інформації чи просто роздумів, а враховуючи, що ММ побудовані таким чином, що згори знаходяться чати у яких були останні повідомлення, тобто свіжіші чати завжди згори, то чати для обговорення якихось питань спускатимуться вниз і користувачі будуть витрачати багато часу на пошук потрібного чату.

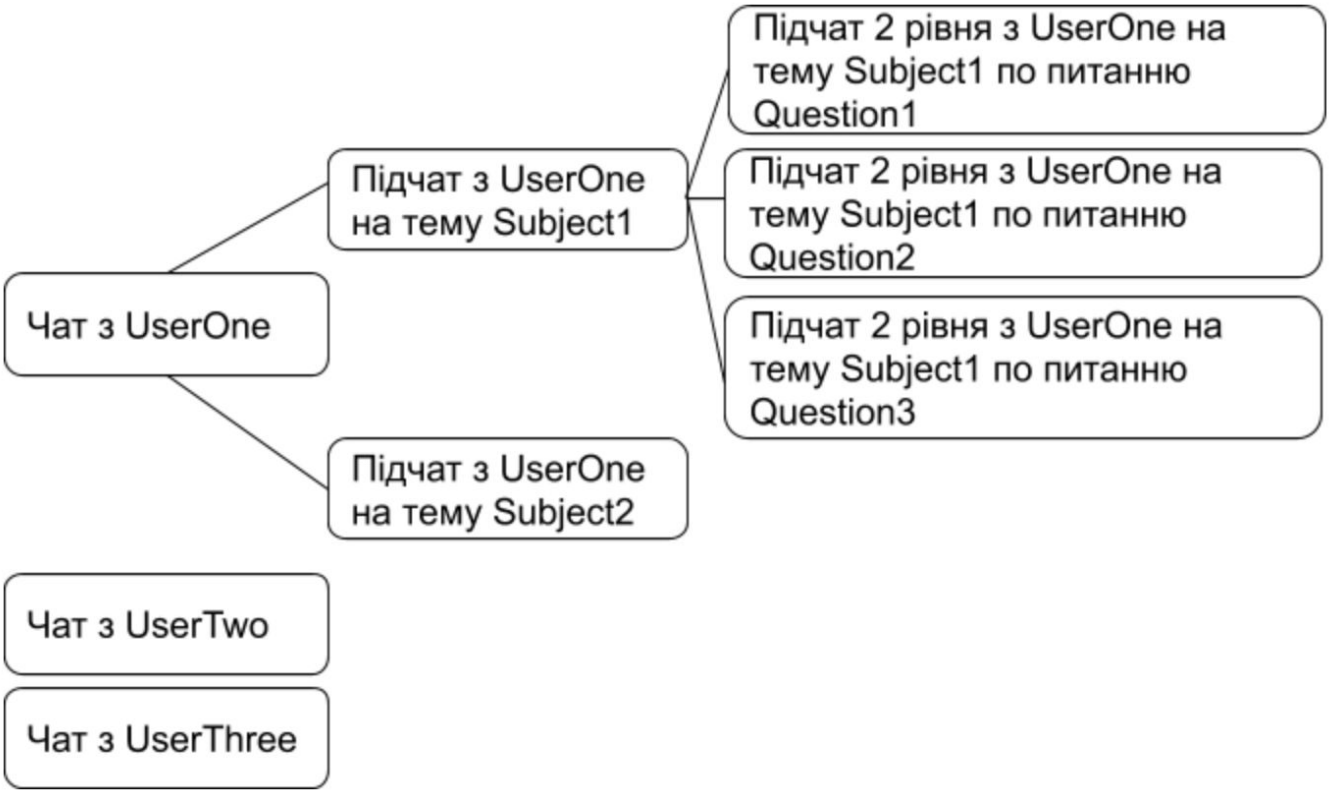


Рисунок 2.2 – Загальна схема архітектури деревопобідного чату.

2.1 Вибір платформи для побудови крос-платформного мобільного додатку

У 2018 році ринок крос-платформних додатків досяг 7,5 мільярдів доларів, а кількість інструментів для розробки мобільних платформ зростає щороку [15]. Кожна платформа має свої переваги на недоліки. У додатків, що пишуться під iOS, Android та веб окремо є виграш у швидкодії та оптимізації роботи, але складність розробки та підтримки, а також вартість такого рішення часто нівелює ці плюси. Написаний один раз кросплатформений додаток може бути доступним одразу користувачам, що використовують різні мобільні операційні системи, чи взагалі доступним за допомогою браузеру в мережі інтернет. Саме тому крос-платформний підхід отримав широкий розвиток та продовжують з’являтися нові рішення як на базі web технологій так і на базі графічних рушіїв.

Графічні платформи вже давно використовуються не лише для створення різноманітних 2d-3d ігор. За рахунок того, що ігри розробляються для всіх мобільних платформ і часто для вебу одночасно, графічні платформи легко імплементують кросплатформну реалізацію без додаткового втручання. Вони є чудовою альтернативою при створенні графічних, і не тільки, застосунків і особливо для прототипування.

Xamarin - це одна з найпопулярніших та найстаріших платформ для розробки додатків на платформі з C #. Xamarin був випущений в 2011 році і придбаний корпорацією Microsoft та підходить для побудови більшості типів мобільних додатків, оскільки технологія поділена на три різні частини: Xamarin Platform, Xamarin Insights та Xamarin Cloud. Цей фреймворк є безкоштовним для невеликих команд до 5 чоловік.

Xamarin надає повну підтримку апаратних функцій, таких як GPS, камера тощо. Більше підходить для побудови додатків із простою графікою, також додатки мають великий розмір та потребують більше часу для розгортання[16].

React Native - це фреймворк з відкритим кодом, створений компанією Facebook у 2015 році. Він дозволяє користувачам використовувати мову JavaScript та React разом із можливостями власної платформи Native API для створення мобільних додатків.

Проте великим недоліком для нас буде те, що для створення веб застосунку треба використовувати додатково ReactJS, і хоч вони мають однаковий синтаксис це все одно ускладнює прототипування.

Solar2D - це безкоштовний набір крос-платформних програм для розробки програмного забезпечення з відкритим кодом, розроблений Corona Labs Inc. у 2009. Він дозволяє створювати мобільні додатки для iOS, Android та Kindle, десктопні програми для Windows, Linux та macOS, а також підключені телевізійні програми для Apple TV, Fire TV та Android TV. Solar2D

Solar2D використовує скриптову мову Lua, що за ідеологією та реалізацією подібна до JavaScript. Lua також реалізує прототипну модель ООП, проте має синтаксис схожий на Pascal. Характерною особливістю Lua є реалізація великого числа програмних сутностей за допомогою мінімуму синтаксичних засобів. Так масиви, структури, множини, черги та списки реалізуються через механізм таблиць, а механізми об'єктно-орієнтованого програмування, включаючи множинне успадкування - з використанням метатаблиць.

В даній роботі буде використовуватися платформа Unity через те, що вона дійсно пропонує створювати додатки, що працюють на більш ніж 25 різних платформах, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-додатки та інші, а також використовує C#, як мову програмування [17].

Unity був створений у 2005 році і до 5 версії підтримував написання коду на Python-подібній мові Boo, також до 2017 року існувала версія UnityScript, що підтримувала JavaScript, від якої теж відмовились на користь C# [18]. Все ще разом з

популярністю та великою кількістю бібліотек робить Unity чудовим інструментом для прототипування будь-яких додатків.

2.3 Вибір платформи для побудови серверної частини мобільного месенджера

Для бекенду ММ нам знадобиться серверне рішення. Ми можемо використати декілька підходів для вибору розміщення серверу:

- розміщення серверу на власному комп'ютері
- розміщення серверу у дата центрі
- використання Cloud сервісу

Складно знайти переваги для розміщення серверу на власному комп'ютері крім отримання навичок налаштування та цілодобової підтримки безперервної роботи серверу. До недоліків можна віднести:

- наявність окремого комп’ютеру для серверу.
- постійний моніторинг та обслуговування серверу.
- швидкість і надійність роботи серверу буде обмежена домашнім провайдером інтернету.

При оренді фізичного сервера у хостинг-провайдера сервер розташований на технічному майданчику хостинг-провайдера, постійно підключений до безперебійної мережі електроживлення і високошвидкісних каналів передачі інформації. До мінусів такої системи можна віднести одразу великі витрати на оренду, складність у розрахунку мінімального розміру серверу та складності у подальшому масштабуванні.

Cloud платформа - це сукупність послуг, програм та обладнання (тобто віддалених серверів), які інші компанії можуть використовувати для вдосконалення власних послуг. Сюди входять такі речі, як управління базами даних, Cloud сховище, послуги IoT, безпека тощо. Це іноді називають "Інфраструктура як

послуга". Cloud платформи надають компаніям масштабовані рішення для зберігання, отримання та управління даними. У свою чергу, вони можуть маштабуватися, щоб задовольнити запити користувачів. Це життєздатний варіант для підприємств будь-якого розміру завдяки ціновим планам, які стягують з клієнта плату за користування інфраструктурою по факту використання.

Cloud сервіс не має описаних вище недоліків, а також пропонує багато початкових послуг безкоштовно або з правом безкоштовного доступу перший рік користування. Саме тому ми використовуємо Cloud платформу.

Декілька переваг Cloud архітектури

- допомагає зменшити витрати, бо не доведеться оновлювати систему, купувати нове програмне забезпечення, платити експертам або витрачати гроші на споживання енергії на обслуговування сервера.

- швидкість роботи та автоматичні оновлення.

- легке та динамічне масштабування. Наприклад, у годину пік можна додати апаратне забезпечення, а також вилучити його вночі, коли ці ресурси не потрібні.

На сьогодні трійка лідерів з впровадження хмарних рішеньскладається з Amazon Web Services, Microsoft Azure та Google Cloud Platform. Завдяки глобальній пандемії, віддаленої роботи та економічної кризи у 2020 році компанії все більше будуть переходити до хмар.

AWS є дочірньою компанією amazon.com, яка надає платформу хмарних рішеньна вимогу приватним особам, компаніям та урядовим організаціям на основі платних підписок. Він один із найстаріших хмарних провайдерів, публічно випущений в 2006 році [19].

Послуги, що пропонує AWS включають аналітику, AR та VR, інтеграцію додатків, блокчейн, обчислення великих даних, маркетингова аналітика, інструменти для розробників, машинне навчання, запуск віртуальних машин, IoT, мобільні додатки, службу робототехніки. Загалом AWS пропонує більше 175 послуг

та охоплює 77 зон доступності в 24 географічних регіонах по всьому світу. Найближчим часом планується створити ще 12 зон доступності та 4 регіони[20].

До мінусів можна віднести складність самої системи за рахунок її масштабності та необхідність підготовки для роботи з екосистемою, AWS також вимагає багато роботи з налаштування, чого не можна сказати про інші хмарні сервіси, що надають рішення "з коробки"[20].

AWS користуються такі компанії як Netflix, Airbnb, Unilever, BMW, Samsung, MI, Zynga та інші [19]

Microsoft Azure, який спочатку називався Azure, був запущений у 2010 році з метою забезпечити компетентну платформу хмарних рішень для бізнесу. Azure було перейменовано на «Microsoft Azure» у 2014 році, хоча назва «Azure» все ще широко використовується. Microsoft Azure пропонує майже той самий перелік функцій та послуг, що і AWS. Платформа надає можливість розгортати віртуальні машини та маштабувати їх у будь-який спосіб. Це дозволяє обробляти та обчислювати протягом декількох хвилин і на будь-якій потужності. Як AWS, так і Azure обробляють програмне забезпечення, яке запускає широкомасштабні паралельні пакетні обчислення, що є перевагою над Google Cloud Platform. На сьогодні налічується 60 зон доступності [21].

Однак є деякі додаткові функції, що є характерними лише для Azure:

- Стійкість даних.

Хмарне сховище Azure захищає дані в центрах обробки даних Microsoft. Будь-яка інформація, за замовчуванням має ще три копії, що унеможливлює втрату чи пошкодження даних[21].

- Захист даних.

Azure шифрує дані кількома способами. За замовчуванням усі служби зберігання Azure увімкнені для шифрування на стороні сховища (SSE), яке використовує 256-бітове шифрування AES. Azure Key Vault дозволяє клієнтам використовувати власні ключі для шифрування[21].

- Легка інтеграція з додатками Microsoft,

З іншими послугами, що надаються корпорацією Microsoft (наприклад, .NET).
До мінусів можна віднести слабку підтримку інших операційних систем та можливі проблеми з каналом, для компаній, що знаходяться не у списку регіональних зон[22].

Google Cloud Platform

Google Cloud Platform (GCP), який пропонує Google, - це набір служб хмарних рішень, що працює на тій самій інфраструктурі, що Google використовує внутрішньо для своїх кінцевих продуктів, таких як пошукова система Google, YouTube тощо

Google Cloud Platform розпочав свій шлях у 2011 році, і менш ніж за десятиліття йому вдалося зайняти істотну частину ринку хмарних рішень.

Окрім версій серверів Windows, Google підтримує кілька поколінь Linux і має кілька служб для девелоперів, таких як, наприклад, App Engine [22]. До переваг можна віднести

- Хмарне сховище.

Google Cloud Storage використовує надшвидкі постійні диски в порівнянні з іншими хмарними сховищами даних та пропонує постійне зберігання об’єктів. Технології Hadoop та Big Table, запроваджені Google, такі як Big Query, Big Table, Hadoop, Big Query, інтегровані та повністю підтримуються. Це є принциповим для тих проектів, які використовують машинним навчанням або аналітикою великих даних[22].

Одним з найбільших недоліків є те, що платформа Google Cloud - це бізнес B2C, і великі підприємства вважають складним працювати з їхніми хмарними службами. Проте для нашого прототипу, це є скоріше плюс ніж мінус.

Ми вибрали GCP App Engine тому що він найкраще підходить для маленьких проектів і провайдер надає 300 USD які можна витратити на сервіси [22].

Для застосунку нам треба - бекенд сервер з зовнішнім ір з HTTPS та базою даних.

Мовою програмування був обраний Python, через легкість налаштувать упорівнянні з Java [23][24].

2.4 Вибір бази даних для мобільного додатку

Для початку ми порівняємо NoSQL та SQL бази даних, це великі групи баз даних які можуть використовуватись для нашого рішення.

SQL розшифровується як "Structured Query Language", або Структурована мова запитів. Це програмна мова, створена для маніпулювання даними в реляційних базах даних. Саме тому часто реляційні бази даних називаються за назвою мови, за якою з ними можна працювати, тобто SQL. Під SQL базами даних ми розуміємо саме реляційні бази даних.

NoSQL розшифровується як "Not only SQL", або "Не тільки SQL". Причому часто він розшифровується як "No SQL", або "Без SQL", але ця розшифровка є помилковою. Такою похідною назвою називають групу реляційних баз даних.

Головна відмінність між ними у тому, що реляційні бази даних зберігають структуровані дані, що як правило являють собою об'єкти реального світу (дані про людину, товари у корзині магазину), згруповані у таблицях, формат яких є заданим на етапі створення бази даних.

Нереляційні ж побудовані інакше. Наприклад, документо-орієнтовані бази зберігають інформацію у вигляді ієрархічних структур даних. Мова може йти про об'єкти з випадковим набором атрибутів. Те, що у реляційній базі даних може бути розбито на декілька взаємопов'язаних таблиць, в нереляційній базі даних може зберігатися у вигляді цілісного об'єкту.

Нереляційні бази з'явилися та стали популярними за рахунок своєї швидкодії, гарного масштабування, що особливо важливе у хмарних базах даних, та легкості використання, оскільки не потребують проектування на етапі створення й використання, й можуть зберігати неструктуровані дані. При чому у ході роботи вона може зберігати нові об'єкти даних без додаткових зусиль.

До переваг SQL баз даних можна віднести відповідність до принципів ACID (атомарність, відсутність протиріч, ізольованість, довготриваліть). Це дозволяє зменшити вірогідність несподіваної поведінки системи й забезпечує цілісність бази даних. Також до плюсів можна віднести структурованість даних, що дозволяє краще розділяти функції зберігання даних, та роботи з ними.

У нашому випадку вимоги до бази даних на етапі прототипу достатньо малі, тому ми можемо використати будь яку базу даних. Однак маючи більшій досвід роботи з SQL базами даних, ми будемо використовувати саме їх.

З SQL баз даних можна виділити дві найбільш популярні, це MySQL та Postgres.

MySQL була створена незалежною компанію, як альтернатива БД mSQL. Але у 2008 її придбала Oracle. З придбанням компанії використання бази даних було змінено на 2 моделі: безкоштовно за ліцензією GPL, тобто з розкриттям власного коду, або з оплатою комерційної ліцензії.

Postgresql на відміну від MySQL не має таких обмежень.

Щодо функціоналу обох баз даних, то для наших цілей обидві бази даних мають достатньо функціоналу.При цьому PostgreSQL виявився більш стабільним та швидким рішенням.Тому ми обрали як рушій бази даних, саме PostgreSQL.

2.5 Висновки до розділу

В даному розділі проаналізовано можливі технології для розробки клієнт та back-end частин прототипу ММ. Для побудови клієнтської частини обрано ігровий рушій Unity з написанням скриптів на мові C# у середовищі розробки Microsoft Visual Studio. Для реалізація серверної частини обрано хмарну інфраструктуру, а саме GCP через їх службу App Engine. Серверною мовою обрано Python, фреймворк - Flask. Базу даних обрано PostgreSQL.

3 Реалізація програмного забезпечення мобільного месенджера

Для імплементації додатку ми використаємо наступні технології, що представлені на рисунку 3.1.

Клієнтська частина ММ написана за допомогою графічного рушія Unity. Для написання back-end частини використовувався Python Flask, для функціонування якого використовується App Engine. Дані користувачів зберігаються у PostgreSQL, яка розміщена у Cloud SQL. Back-end частина та база даних функціонують на базі GCP.

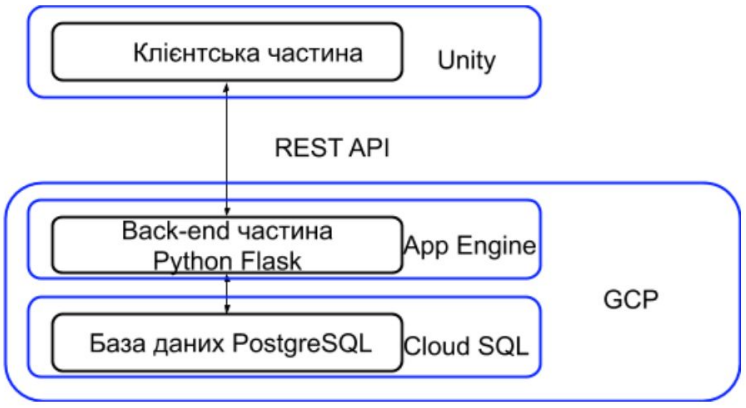


Рисунок 3.1 – Архітектура мобільного месенджера

3.1 Клієнтська частина розробки мобільного месенджера

3.1.1 Архітектура клієнтської частини мобільного месенджера

Для архітектури клієнтської частини (рис 3.2) ми використовуємо модель MVC Model-View-Controller - це паттерн у розробці програмного забезпечення, який зазвичай використовується для реалізації користувацьких інтерфейсів, даних та логіки управління. Він підкреслює поділ між бізнес-логікою програмного

забезпечення та екраном дисплею. Це "розділення проблем" забезпечує кращий розподіл праці та поліпшення обслуговування. [25]

Рисунок 3.2 – Архітектура клієнтської частини мобільного месенджера

- 3.1.2 Огляд побудови UI клієнтської частини мобільного месенджера

Розглянемо як виглядає логін сторінка користувача. На рисунку 3.3 зображено поля де користувач вводить свій логін та пароль, при натисканні на зелену кнопку Login відбувається доступ користувача до ММ

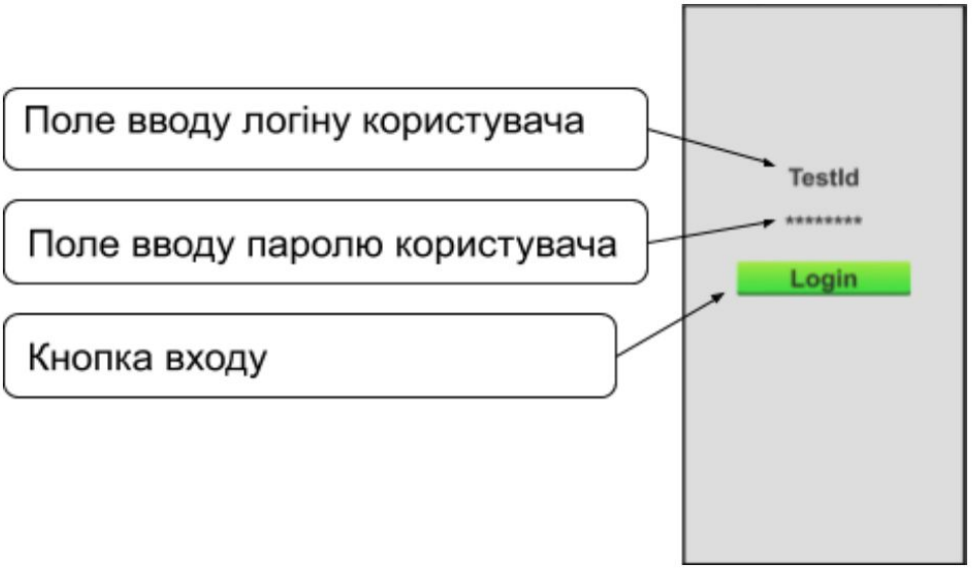


Рисунок 3.3 – Копія екрану логіну

Після коректного логіну користувач пострапляє на керан, що містить всі розпочаті чати (рис 3.4).

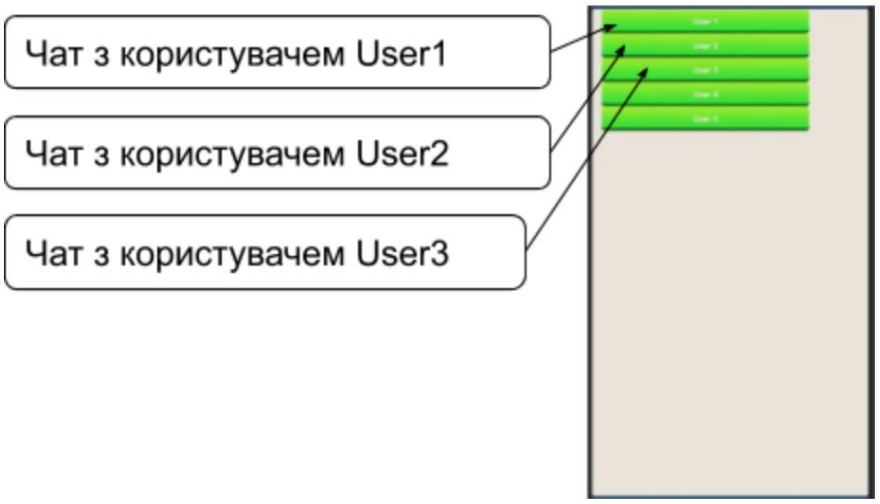


Рисунок 3.4 – Копія екрану з переліком чатів з користувачами

При тапі на будь-якому з чатів відкривається вікно, що зображене на рисунку3.5. Зверху кнопка “Додому” - що завжди повертає у загальний список чатів, кнопка “Назад” - що повертає на один екран назад, тобто з підчату в головний чат або з головного чату у загальний список чатів. З лівої сторони назва загального чату.

Далі йдуть підчати першого рівня, в даному випадку їх 3. За рахунок того, що структура підчату деревоподібна (Рис 2.2) в загальному чаті зображені лише підчати першого рівня.

В самій нижній частині екрана є вікно для вводу користувачем тексту та кнопка щоб відправити повідомлення.

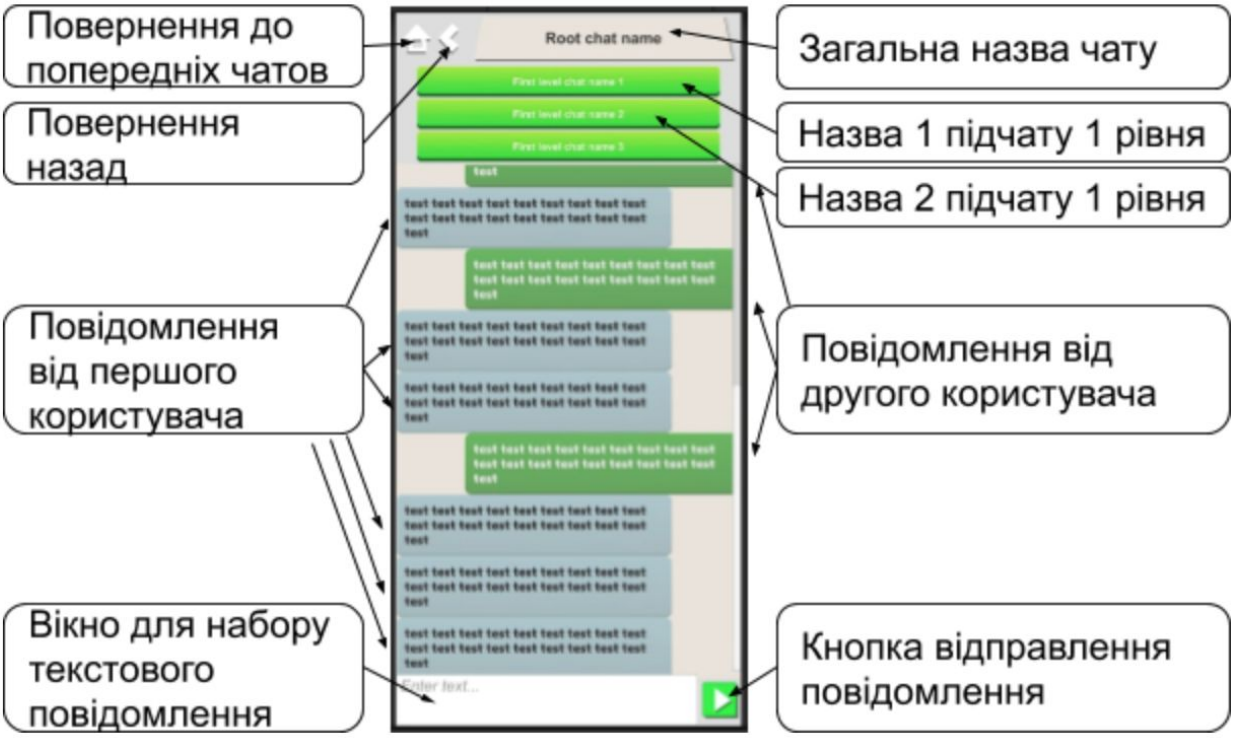


Рисунок 3.5 – Копія екрану чату з окремим користувачем

На рисунку 3.6 зображено зовнішний вигляд підчату третього рівня з кнопкою Branch. Ця кнопка відповідає за створення нового вкладеного підчату. Повідомлення на якому вона буде натиснута скопіюється та стане першим у новому підчаті 4 рівня.

Жовтий трикутник, що зображений внизу екрана інформує користувача, що у нього є непрочитані повідомлення в поточному чаті.

Рисунок 3.6 – Копія екран підчату третього рівня

3.1.3 Реалізація класів відповідно до екранів інтерфейсу (Views)

Реалізація сторінки логіну користувача

Для реалізації цього екрану дотримуючись концепції MVC ми використали декілька вбудованих у рушій UI елементів інтерфейсів користувача: поля для вводу логіну та паролю та кнопка Login. Ми імплементуємо LoginBtnControler клас (рис 3.7), який буде отримувати значення з полів тексту для введення, перевіряти їх на допустимість значень та відправляти значення в клас-прошарок для роботи з мережею. У випадку коректної ідентифікації клас викликає метод LoginIsOk і переключає аплікацію на екран побудови чатів.

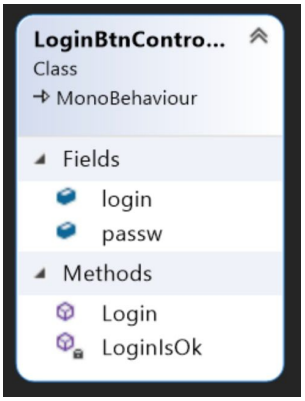


Рисунок 3.7 – Клас LoginBtnControler

Екран чату користувача

На цьому екрані користувач бачить всі розпочаті доступні чати з іншими користувачами. Для побудови цього екрану ми використовуємо вбудований елемент інтерфейсу користувача ScrollView. Цей елемент дозволяє нам легко будувати динамічні списки елементів інтерфейсу.

Цей екран реалізується за допомогою двох класів контролерів ChatListViewer та ChatViewController (рис 3.8). Клас ChatListViewer отримує всі кореневі чати, які доступні для користувача з класу прошарку збереження даних. З отриманих даних він будує індивідуальні елементи (ChatView). Також ChatListViewer раз в секунду перевіряє клас-прошарок для збереження даних на наявність чатів с новими користувачами. Кожний з побудованих елементів керується окремим класом контролером ChatViewController. Індивідіальні елементи слугують також кнопками при натиску на будь-яку з них відкривається екран індивідуального чату між користувачами.

Для зберігання і передачі інформації про існуючі чати ми створюємо модель класу Chat

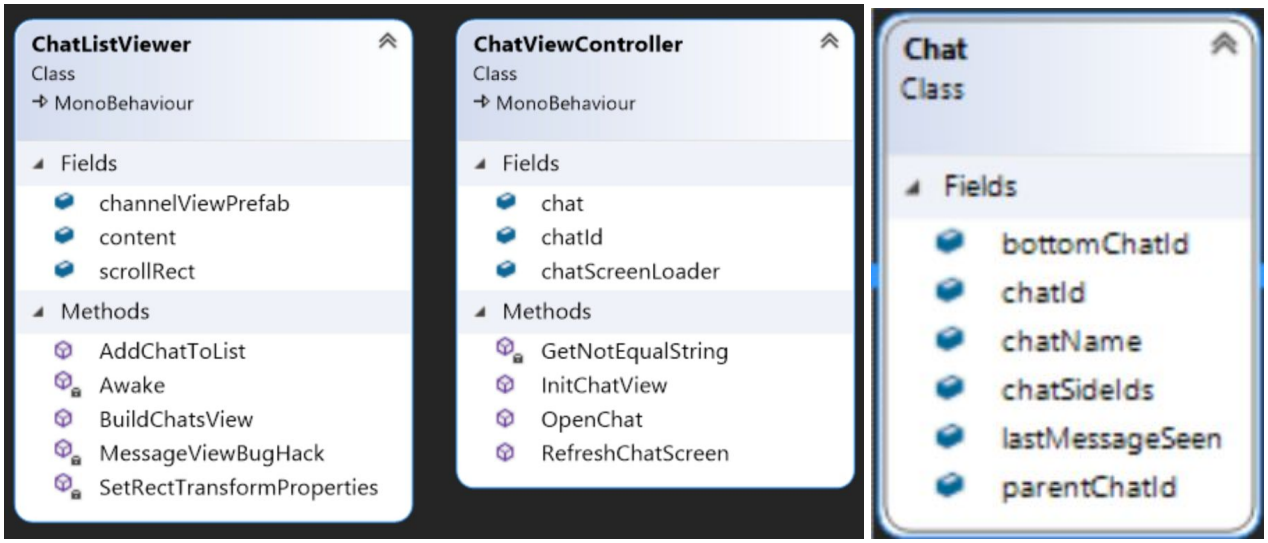


Рисунок 3.8 – Класи ChatListViewer, ChatViewController та ChatListViewer та Chat

Екран перегляду повідомлень між учасниками чату, вкладених чатів, назви чату та поле для введення власного повідомлення у чат

Оскільки екран містить багато елементів та контролерів ми почнемо його розглядати за принципом згори донизу. Почнемо з класу який керує всім цим екраном - ChatScreenLoader (рис 3.9). Цей клас виконує головну роботу з ініціалізації та завантаження головного екрану та містить посилання на контролери інших елементів на цьому екрані, ініціалізує їх завантажує дані з прошарку зберігання даних.

Також він опрацьовує натискання кнопок “Назад” та “Додому”. Кнопка “Назад” повертає з на одну ступінь назад, тобто з підчату3 до підчату2, з підчату1 у чат, або з загального чату у список всіх чатів у той час як кнопка “Додому” завжди повертає на загальний список чатів у якому б чаті чи підчаті не знаходився користувач.

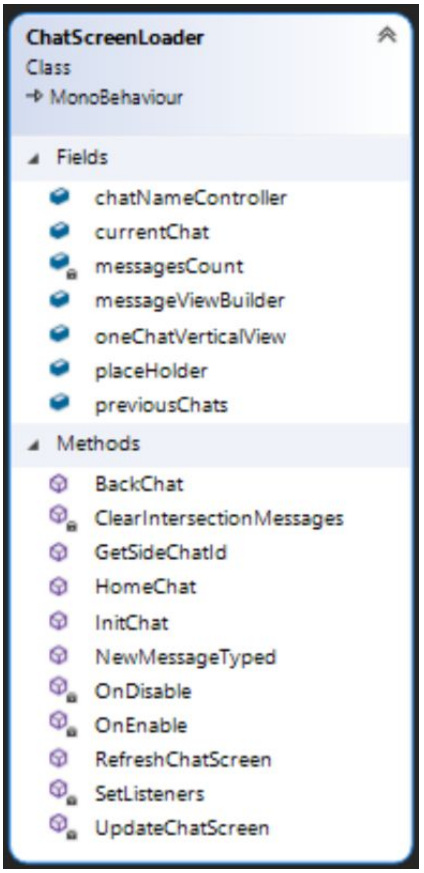


Рисунок 3.9 – Клас ChatScreenLoader

Клас ChatNameController (рис 3.10) це невеликий клас, який відповідає за відображення назви чату в верхній панелі

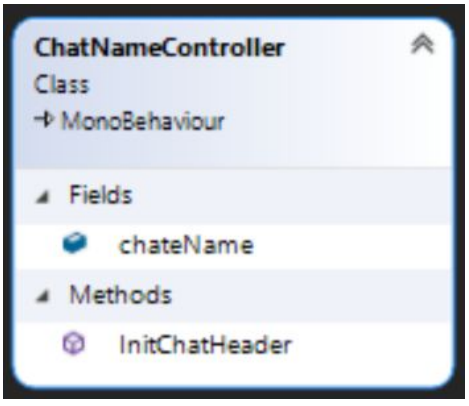


Рисунок 3.10 – Клас ChatNameController

Клас OneChatVerticalView (рис 3.11) відповідає за формування переліку вкладених чатів. Він відображає scroll view де кожна кнопка підписана назвою вкладеного чату. Відповідно кожне назва чату є кнопкою при натисканні на яку користувач переходить у цей чат.

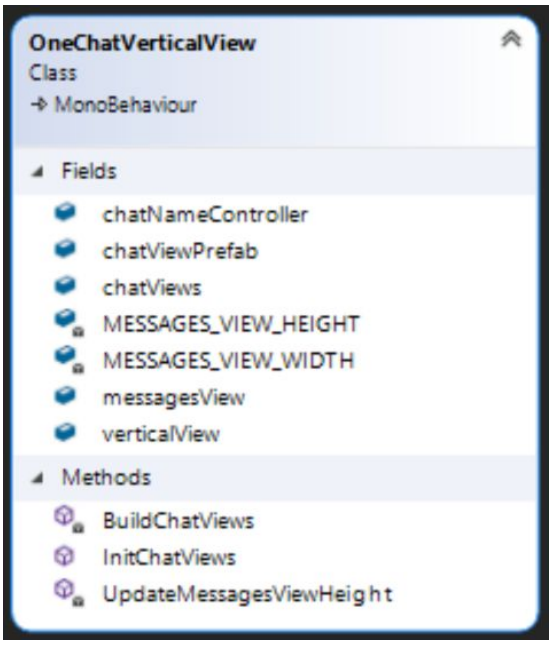


Рисунок 3.11 – Клас OneChatVerticalView

Клас MessageViewBuilder (рис 3.12) будує список повідомлень у чаті. Кожний елемент цього чату являє собою окремий об'єкт який керується класом MessageViewControler. Цей клас відповідає за відображення текстової частини повідомлення, а також керує додатковою кнопкою Branch, що з'являється при натисканні і утриманні текстового повідомлення протягом 3 секунд. Це повідомлення копіюється та стає першим у новому дочірньому чаті який створюється при натисканні на кнопку Branch. Клас Message створений для зберігання інформації про повідомлення в системі у базі даних.

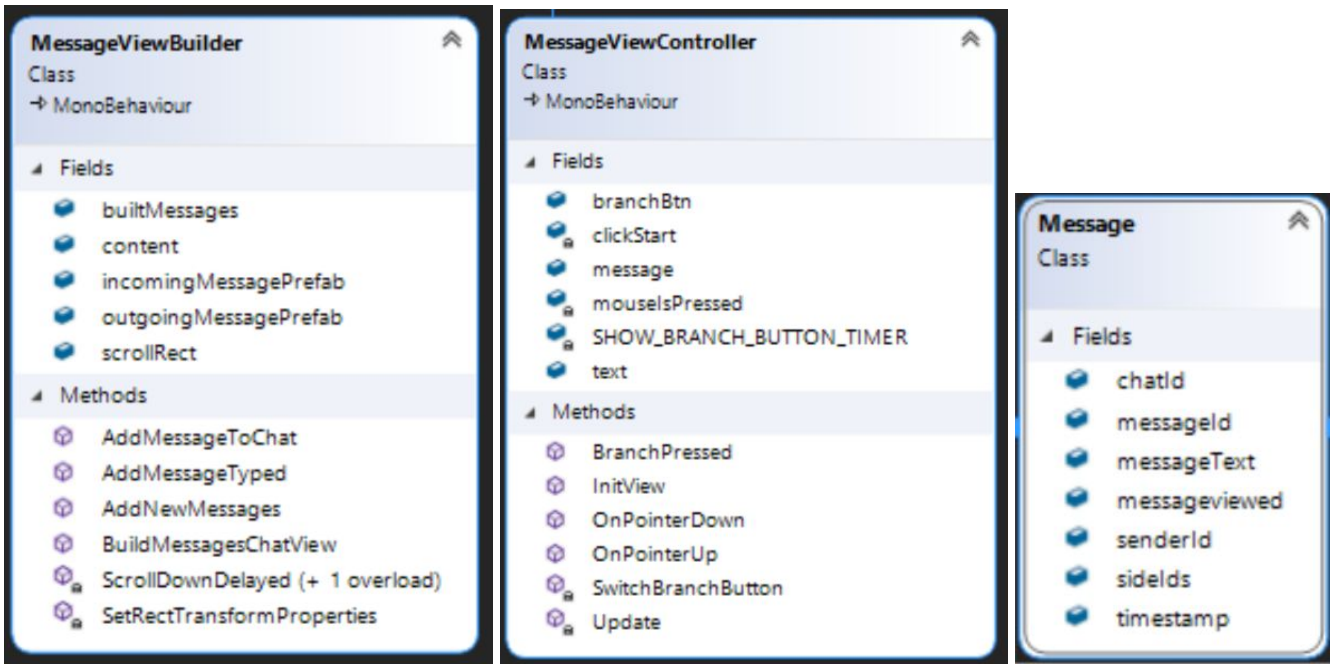


Рисунок 3.12 – Класи MessageViewBuilder, MessageViewControler та Message

Клас SendMessageController (рис 3.13) виконує функцію опрацювання вводу користувача в поле введення повідомлення при натисканні кнопки Send він створює нове повідомлення відправляє його через мережевий клас прошарок та зберігає його в класі прошарку зберігання даних.

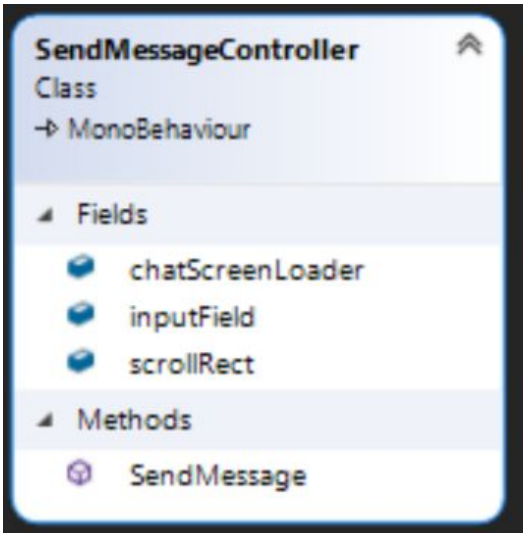


Рисунок 3.13 – Клас SendMessageController

Клас MessageIsViewedController (рис 3.14) виконує просту функцію повідомлення користувачу про наявність непрочитаних повідомлень у поточному чаті. він перевіряє статус кожного повідомлення відображеного у чаті і якщо якесь повідомлення не прочитане він відображає відповідну позначку.

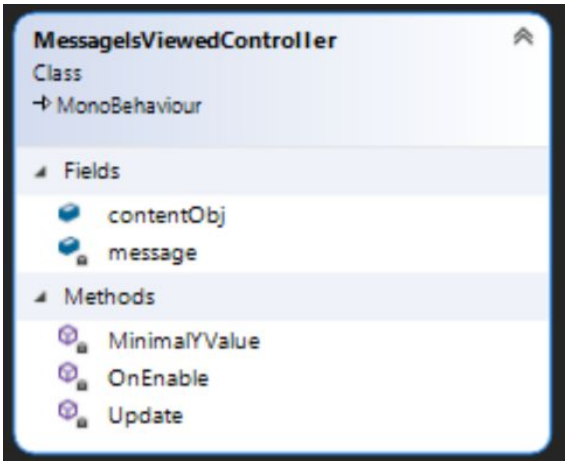


Рисунок 3.14 – MessageIsViewedController

Клас Statics (рис 3.15) це статичний клас, який зберігає загальнодоступним чином ідентифікатор поточного користувача додатку для використання іншими частинами програми.

Рисунок 3.15 – Клас Statics

Клас UIStateMachine (рис 3.16) побудований за патерном одинак(Singlton) є машиною станів (State Machine) графічного інтерфейсу користувача. Він відповідає за перемикання екранів додатку між собою. Тобто з логіну в екран чатів, з екрану чатів до екрану повідомлень та навпаки.

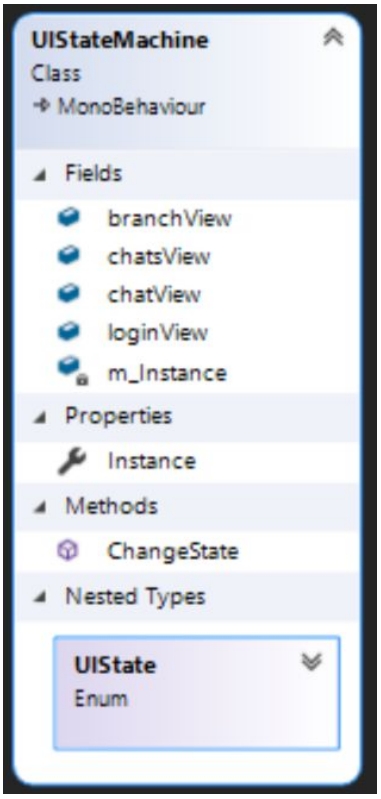


Рисунок 3.16 – Клас UIStateMachine

Клас ChatsHolder (рис 3.17) побудований за патерном одинак (Singlton) є класом прошарком зберігання даних, тобто відповідає за зберігання поточних даних користувача: всіх чатів що веде користувач та всіх повідомлень що були надіслані користувачем або користувачу. Також він відповідає за взаємодію з мережевим класом прошарком: завантаження та оновлення інформації.

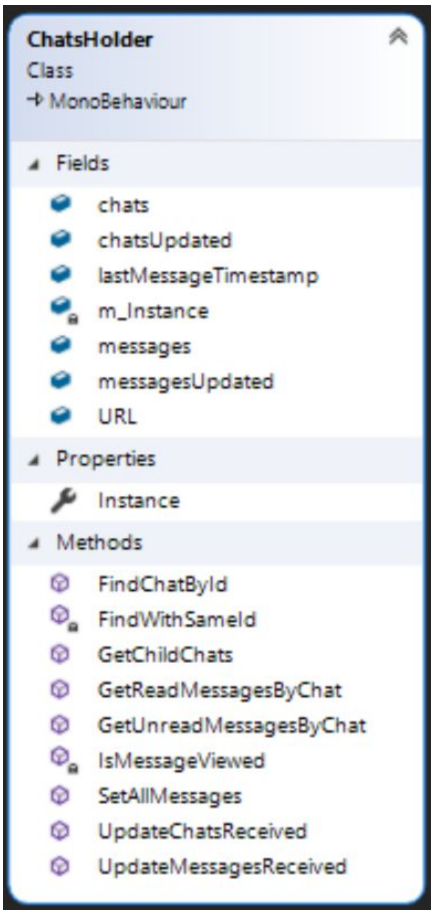


Рисунок 3.17 – Клас ChatsHolder

Клас NetworkingManager (рис 3.18) побудований за патерном одинак (Singlton) є мережевим класом прошарком, тобто він виконує усі функцію зі спідкуванням додатка с backend використовуючи REST API: завантаження початкових даних, оновлення отримання нових повідомлень або чатів, відправлення повідомлень надрукованих користувачем та інше.

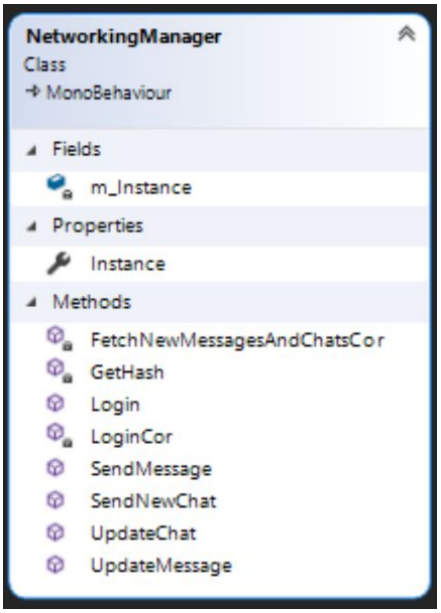


Рисунок 3.18 – Клас NetworkingManager

3.2 Серверна частина мобільного месенджера

Для створення Бекенд Restful API частини ми використали наступний набір технологій: мову програмування Python з фреймворк Flask, що розміщені на Google Cloud Platform App Engine.

Архітектура серверної частини зображена на рисунку 3.19. Ми використали стандартний підхід з використанням RestAPI, бекенду, який опрацьовує запити, та бази даних, що виконує функцію збереження даних користувача.

RestAPI розбитий на групи: операції з користувачами, запити по створенню та оновленню даних та запити по читанню даних.

Кожна точка запиту має свій метод обробки на бекенді, який перевіряє наявність необхідність даних та їх коректність також він виконує відповідні запити до бази даних. Бекенд не зберігає власного стану й за рахунок цього може обробляти будь яку кількість запитів параллельно.

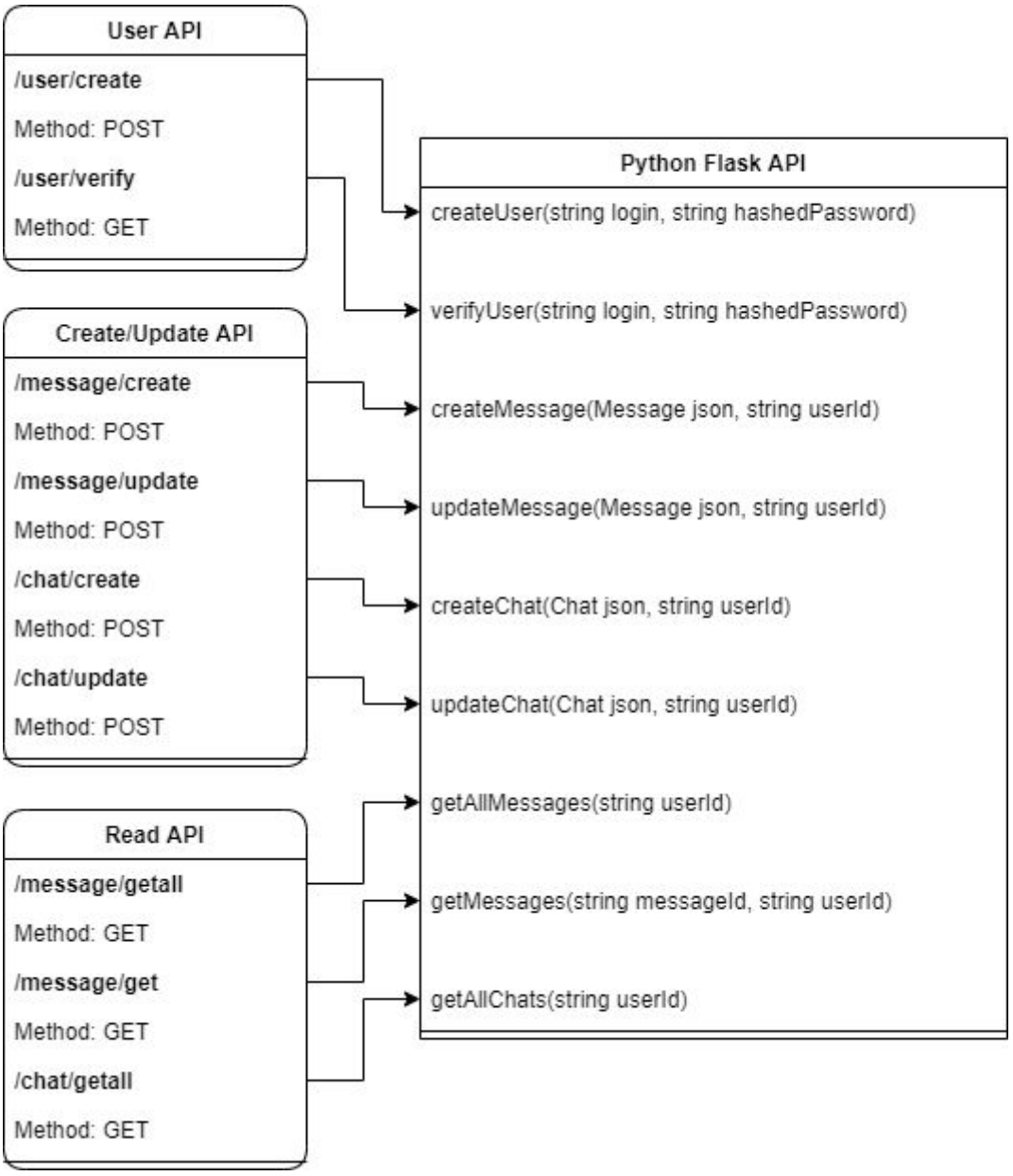


Рисунок 3.19 – Архітектура серверної частини проекту

Процес сетапу App Engine показано на рисунку 3.20. Завдяки GCP все відбувається за декілька кроків. При виборі локації було обрано europe west 3, як найближчий до України.

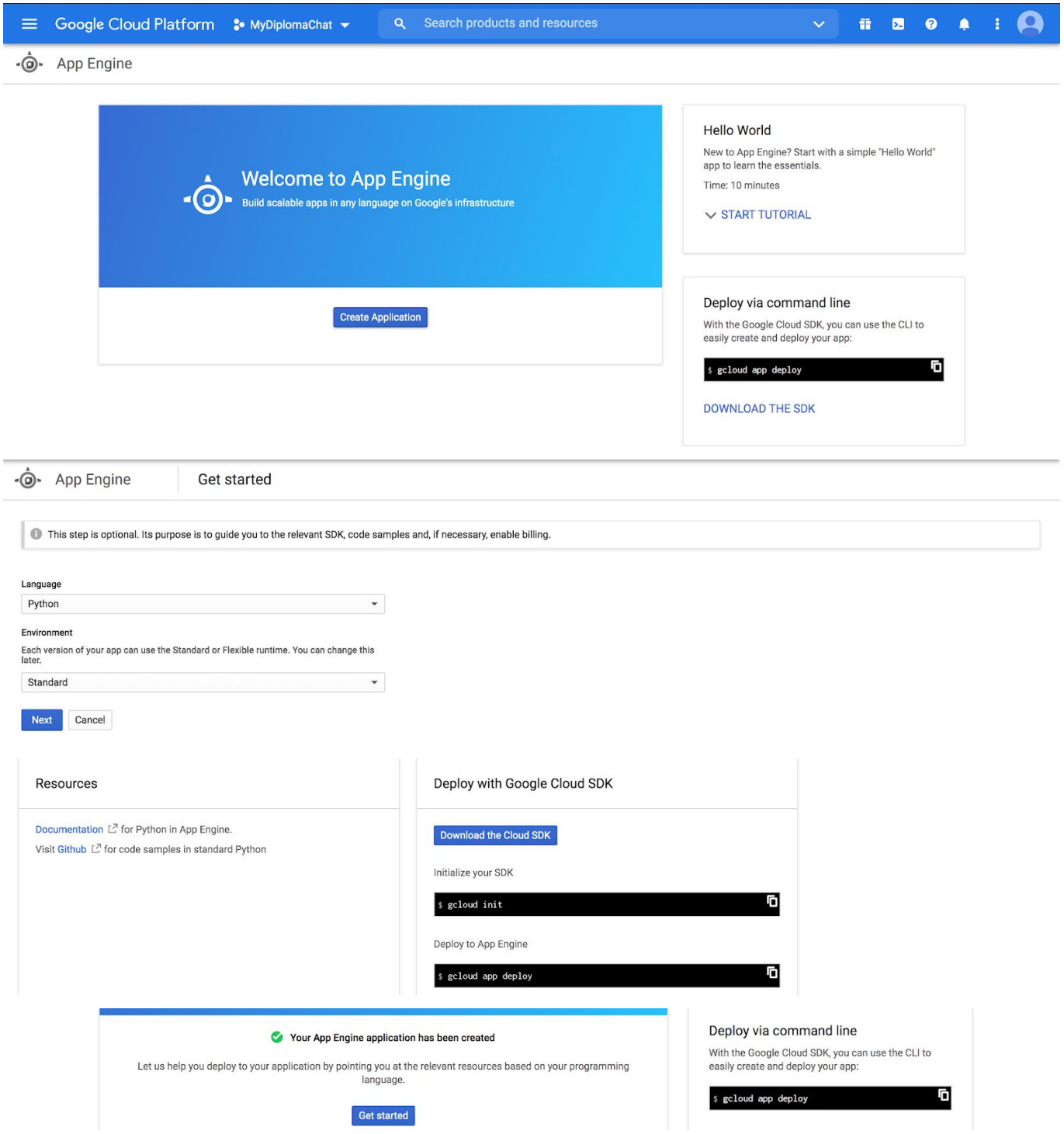


Рисунок 3.20 – Етапи створення App Engine

Restful API було побудовано наступним чином

API для обробки запитів клієнтської частини на створення та оновлення об'єктів.

- API для реєстрації нового користувача

Path: /user/create
Method: POST
Payload: new user login and hashed password json

- API для верифікації нового користувача

Path: /user/verify
Method: GET
Payload: new user login and hashed password json

- API для відправлення нового повідомлення, що написано користувачем у базу даних:

Path: /message/create
Method: POST
Payload: new Message json object
Response: new message timestamp – відмітка часу створення нового повідомлення на сервері

- API для оновлення повідомлення (наприклад для позначення його, як прочитаного користувачем):

Path: /message/update
Method: POST
Payload: Message update json object

- API для створення нового чату (будь якого за глибиною), у випадку коли користувач створює нову гілку чату:

Path: /chat/create
Method: POST
Payload: Chat create json object

- API для оновлення чату (будь якого за глибиною), у випадку коли користувач оновлює назву чату:

Path: /chat/update
Method: POST
Payload: Chat update json object

API для обробки запитів клієнтської частини на отримання інформації.

- API для отримання усіх повідомлень

Path: /message/getall
Method: GET
Payload: user id
Response: усі повідомлення, що були створені, або адресовані користувачу, за весь час

- API для отримання нових повідомлень

Path: /message/get
Method: GET
Payload: user id
Response: усі повідомлення, що були створені, або адресовані користувачу, та які надійшли за час з останнього звернення клієнта

- API для отримання чатів користувача, в яких він є учасником дискусії

Path: /chat/getall
Method: GET
Payload: user id
Response: усі чати користувача, в яких він бере участь

3.3 База даних мобільного месенджера

Архітектура бази даних зображена на рисунку 3.21. База даних є спрощеним варіантом збереження основних даних, що необхідні користувачу: логін, пароль (хешований) користувача, перелік усіх чатів, які є в системі, перелік усіх повідомлень, які є в системі. Для спрощення розробки, ідентифікація чатів та повідомлень користувача виконується за логіном користувача.

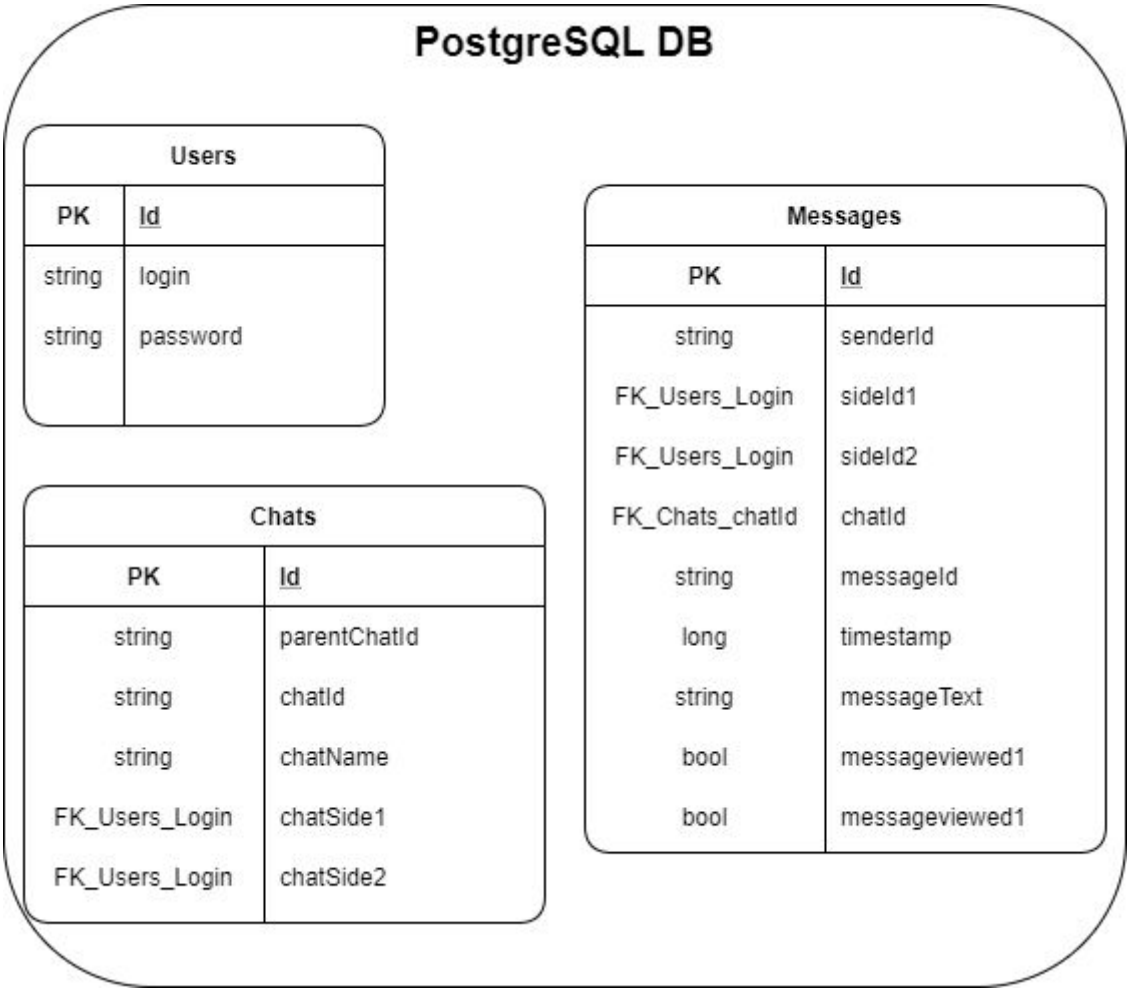


Рисунок 3.21 – Архітектура бази даних мобільного месенджера

Як було викладено вище в даній роботі буде використано PostgreSQL на базі GCP. Всі команди по створенню (рис 3.22) та наповненню таблиць було написано через вбудований у GCP термінал.

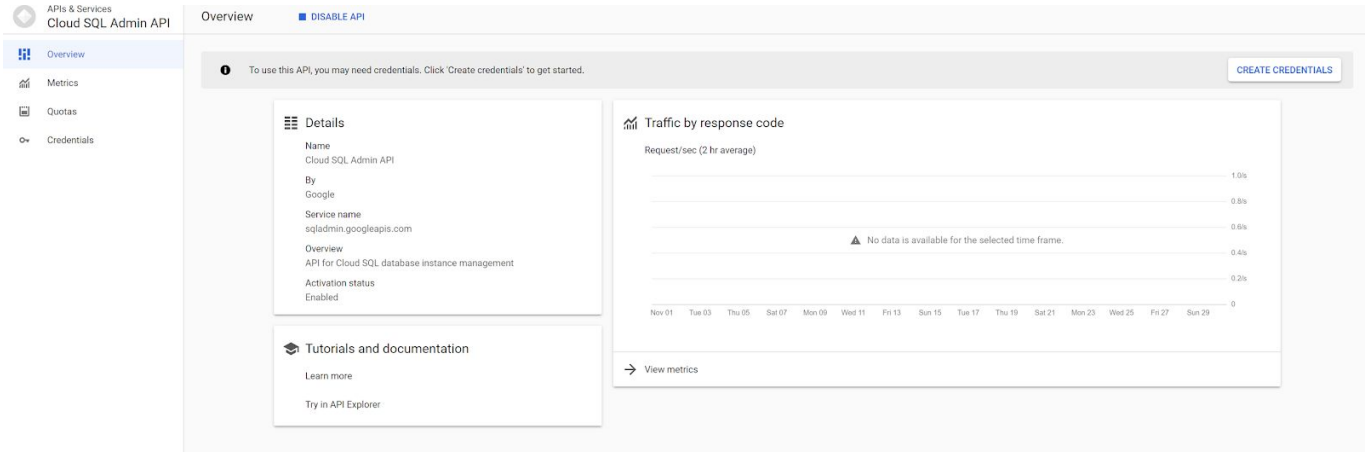


Рисунок 3.22 – Створення бази даних в GCP

Відкриваємо вікно терміналу та підключаємось до бази даних вводячи логін та пароль. На рисунку 3.23 показано що доступ до бази даних відкрито та всі перевірки аутентифікації пройдені.

Рисунок 3.23 – Аутентифікація через терміналі у базі даних

Для роботи в базі даних створимо за домопогою SQL запросів таблицю USERS та введемо для перевірки деякі дані (рис 3.24 та рис 3.25). Ця таблиця буде зберігати логін користувача та хеш його паролю для перевірки ідентифікації якщо

користувач буде входити в систему. Хеш пароль генерувався у Класі NetworkingManager за допомогою методу GetHash.

```
postgres=> CREATE TABLE IF NOT EXISTS "USERS" ("id" serial,"login" text,"password" text, PRIMARY KEY( id ));
CREATE TABLE
postgres=> \dt
         List of relations
 Schema | Name  | Type  |  Owner
--------+-------+-------+----------
 public | USERS | table | postgres
(1 row)

postgres=>
```

```
postgres=> INSERT INTO "USERS"("id", "login", "password") VALUES (2, 'gavrilyuk@gmail.com', '7892cf12810730df1107712904919125ff23ca8ada4d87d1794b3dae47105a3f');
INSERT 0 1
postgres=> SELECT * FROM "USERS";
 id |       login        |                             password
----+--------------------+--------------------------------------------------------------------
  1 | ivanov@gmail.com    | 64cd2ac4788546bd1f2d328a27888dfb44826b2d94aea36963e8253aa9f1b603
  2 | gavrilyuk@gmail.com | 7892cf12810730df1107712904919125ff23ca8ada4d87d1794b3dae47105a3f
(2 rows)

postgres=> INSERT INTO "USERS"("id", "login", "password") VALUES (3, 'sidorenko@gmail.com', 'cba2593afac7cf3ca052dcd0f2ba09b9bb94e0604c4ac2a3f09d2c330bc20442');
INSERT 0 1
postgres=> INSERT INTO "USERS"("id", "login", "password") VALUES (4, 'shevchenko@gmail.com', '176030fde41cc072deba29e1a09414782fb0619877bfb7b9bc98d4e9f054ba49');
INSERT 0 1
postgres=> SELECT * FROM "USERS";
 id |       login        |                             password
----+--------------------+--------------------------------------------------------------------
  1 | ivanov@gmail.com    | 64cd2ac4788546bd1f2d328a27888dfb44826b2d94aea36963e8253aa9f1b603
  2 | gavrilyuk@gmail.com | 7892cf12810730df1107712904919125ff23ca8ada4d87d1794b3dae47105a3f
  3 | sidorenko@gmail.com | cba2593afac7cf3ca052dcd0f2ba09b9bb94e0604c4ac2a3f09d2c330bc20442
  4 | shevchenko@gmail.com | 176030fde41cc072deba29e1a09414782fb0619877bfb7b9bc98d4e9f054ba49
(4 rows)
```

Рисунок 3.24 – Створення та заповнення таблиці USERS

Таблиця CHATS містить інформацію про безпосередньо чати між користувачами, а саме

parentChatID - чат-батько ;

chatId -ідентифікатор поточного чат;

chatName - ім'я поточного чату;

chatSideId1 та chatSideId2 - логіни користувачів які спілкуються у чаті;

lastMessageSeen - дата, коли було переглянуто останнє повідомлення.

Для прикладу, спілкування буде відбуватися у трьох чатах між 4 користувачами.

Робота з таблицею показана на рисунку 3.25.

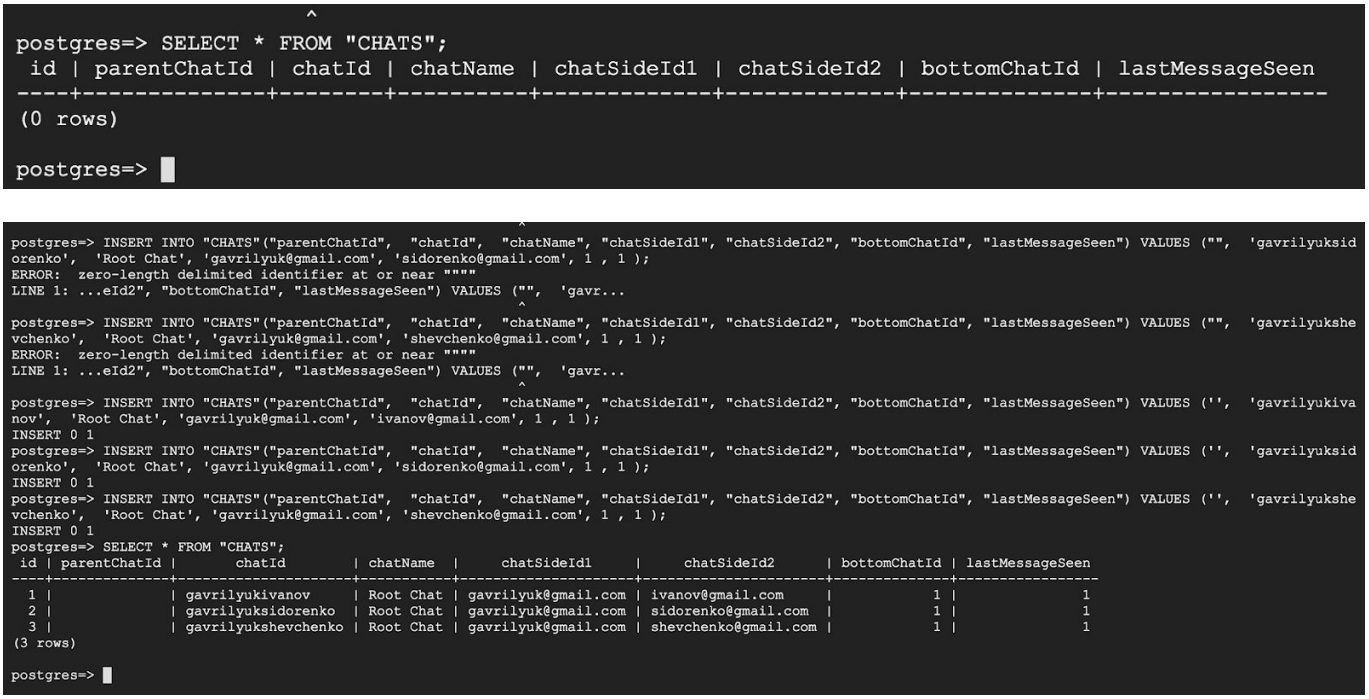


Рисунок 3.25 – Створення та заповнення таблиці CHATS

Таблиця MESSAGES створена аналогічно до двох попередніх таблиць. Також всі таблиці пов’язані між собою.

3.5 Тестування мобільного месенджера

Тестування проведено для веб у браузері Google Chrome (рис 3.26 а), для операційної системи Microsoft (рис 2.26 б) та Android платформи на мобільному телефоні Samsung Galaxy S8 (рис 2.28). Перевірити коректність роботи для MacOS та iPhone не було можливості через відсутність самого обладнання для тестування та через відсутність сертифікації в Apple Developer Center, яка коштує грошей.

На рисунку 2.28 зображено скріншоти екрану мобільного телефону під час користування месенджером. Кнопка Branch, яка відповідає за створення нової гілки чату з’являється після довгого тапу на будь-якому повідомленні, яке потім стає першим у новій гілці співбесіди.

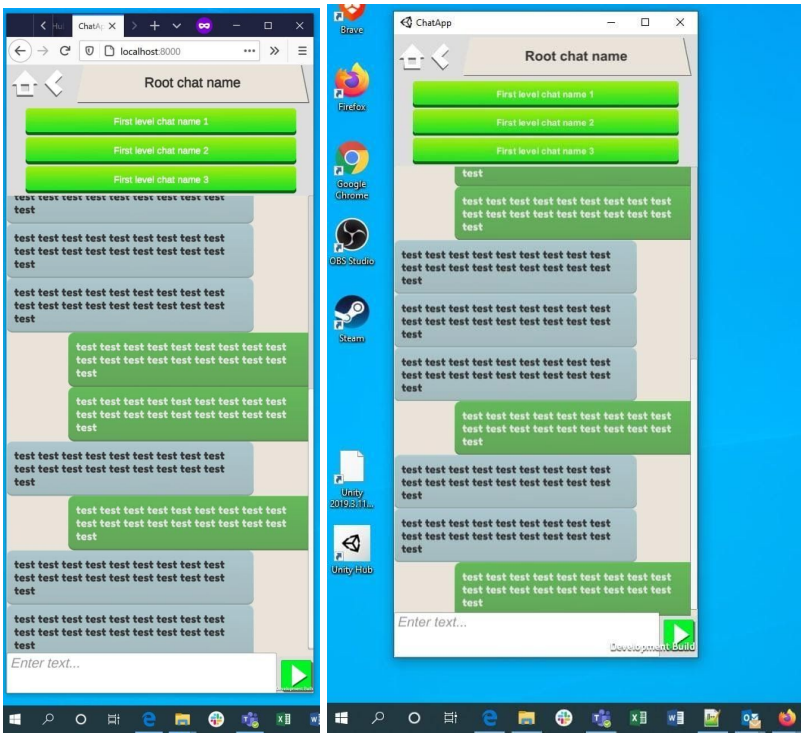


Рисунок 3.26 – Приклад роботи програми у веб браузері (а) та на комп'ютері з операційною системою Microsoft (б)

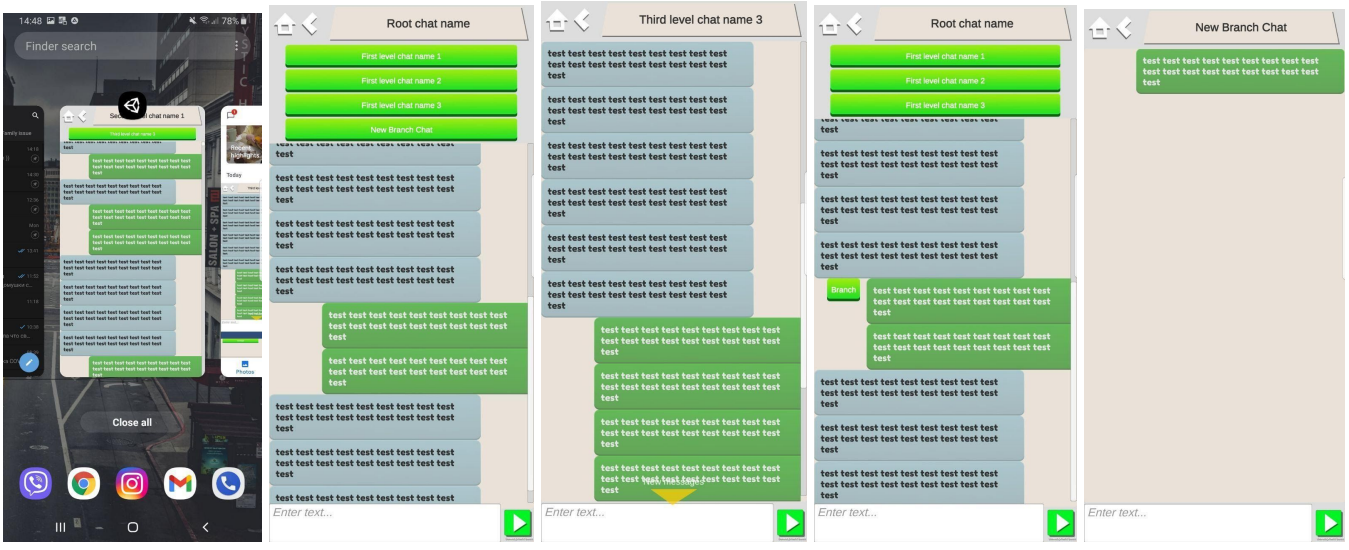


Рисунок 3.27 – Приклад роботи мобільного додатку на телефоні Samsung Galaxy s8. Загрузка застосунку на телефоні (а), приклад основного чату (б), приклад підчату третього рівня з функціоналом нових повідомлень (в), приклад появи кнопки Branch(г), приклад створення нової гілки чатів(д)

3.6 Висновки до розділу

У цьому розділі створено архітектуру ММ з гілкоподібною структурою чатів та реалізовано кросплатформне рішення на базі графічного рушія Unity з серверною частиною написаною мовою Python та фреймворком Flask на GCP з базою даних PostrgeSQL. Серверна частина додатку виконана на платформі CGP App Engine.

Продемонстровано работу застосунку в операційній системі Windows, на мобільному телефоні з системою Android та у веб браузері Google Chrome.

4 Функціонально-вартісний аналіз програмного продукту

У даному розділі проводиться оцінка основних характеристик кросплатформного ММ. Клієнтська частина програмного продукту була створена на мові програмування C#, а серверна на Python. Розробка проводилась у середовищі Microsoft Visual Studio.

Продукт призначено для використання на мобільних девайсах та веб платформі.

4.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод функціонально-вартісного аналізу для проведення техніко-економічного аналізу розробки.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії в кореляції з його користю. Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати [26].

Технічні вимоги до продукту наступні:

- програмний продукт повинен функціонувати на мобільних телефонах з iOS та Android та у web;
- забезпечувати стабільну роботу, а при виникненні помилок в процесі роботи програми надавати розробнику інформацію про їх походження, для подальшого усунення;
- забезпечувати зручність і простоту взаємодії з користувачем;
- передбачати мінімальні витрати на впровадження програмного продукту.

Обґрунтування функцій програмного продукту для клієнтської та серверної частин. Головна функція F_0 – розробка програмного продукту, який аналізує процес за вхідними даними та будує його модель для подальшого прогнозування. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F1 – середовище виконання бекенд коду;

F2 –інтерфейс користувача;

F3 – мова програмування бекенду;

Кожна з основних функцій може мати декілька варіантів реалізації:

Функція F_1:

a. Google App Engine;
b. AWS Elastic Beanstalk ;

Функція F_2:

a. Unity;
b. Xamarin;

Функція F_3:

a. Python;
b. Java;

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис 4.1).

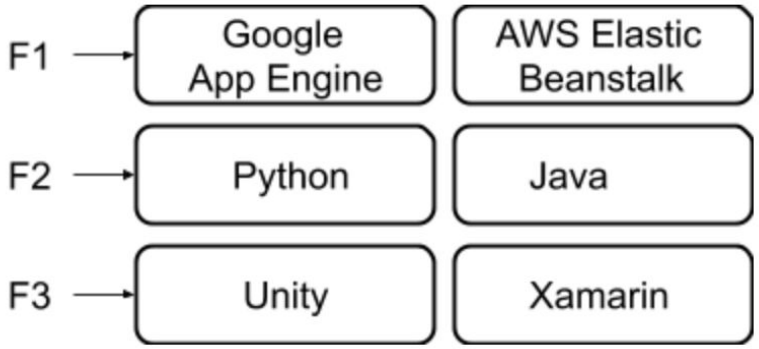


Рисунок 4.1 – Морфологічна карта системи

На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій, які представлені у таблиці.

Ф-ї	**Вар-ти**	**Переваги**	**Недоліки**
F1	a	Легка у налаштуванні	Має широку інфраструктуру
	b	Велика кількість сервісів	Платна
F2	a	Підтримує більше платформ	Орієнтована для створення графічних додатків
	b	Краще підходить для розробки ентерпрайз проектів	Треба окремо прописувати ММ для вебу.
F3	a	Багато бібліотек	Потребує додаткових ресурсів
	b	Краще підтримка багатопотоковості	Мало бібліотек

Таблиця 4.1 – Позитивно-негативна матриця варіантів основних функцій

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, як такі, що не відповідають поставленим перед програмним продуктом технічним вимогам.

Функція F_1.

Оскільки в межах дипломної роботи ми обмежені у фінансах, тому варіант b можна відкинути

Функція F_2.

В даному випадку ми можемо розглядати лише варіант а, бо проект невеликий і додаткові налаштування для коректної роботи серверної частини призведуть до ускладнення розробки.

Функція F_3

Можна розглянути обидва варіанта а та b.

Таким чином, будемо розглядати такі варіанти реалізаціїї ПП:

1. F1a – F2a – F3a
2. F1a – F2 a – F3b

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2 Обґрунтування системи параметрів ПП

На підставі даних про основні функції, що повинен реалізувати програмний продукт та вимоги до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня [26]. Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

X1 – швидкодія мови програмування;

X2 – об’єм пам’яті, потрібної для програми;

X3 – час обробки запитів користувача;

X4 – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

1) *X1*: параметр, що відображає швидкодію операцій залежно від обраної мови програмування.

2) *X2*: параметр, що відображає об'єм пам'яті в оперативній пам'яті персонального комп'ютера, необхідний для збереження та обробки даних під час виконання програми.

3) *X3*: параметр, що відображає час, який витрачається на дії.

4) *X4*: параметр, який показує розмір програмного коду який необхідно створити безпосередньо розробнику.

Таблиця 4.2 – Основні параметри ПП

Назва Параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	**середні**	**кращі**
Кількість платформ яку підтримує рушій	X1	платформа	1	6	25
Швидкодія бази даних	X2	од процесорів	1	2	4
Час обробки запитів користувача	X3	мс	100	50	10
Потенційний об'єм програмного коду	X4	кількість строк коду	3000	1500	1000

За даними таблиці 4.2 будуються графічні характеристики параметрів рис 4.2– рис. 4.5.

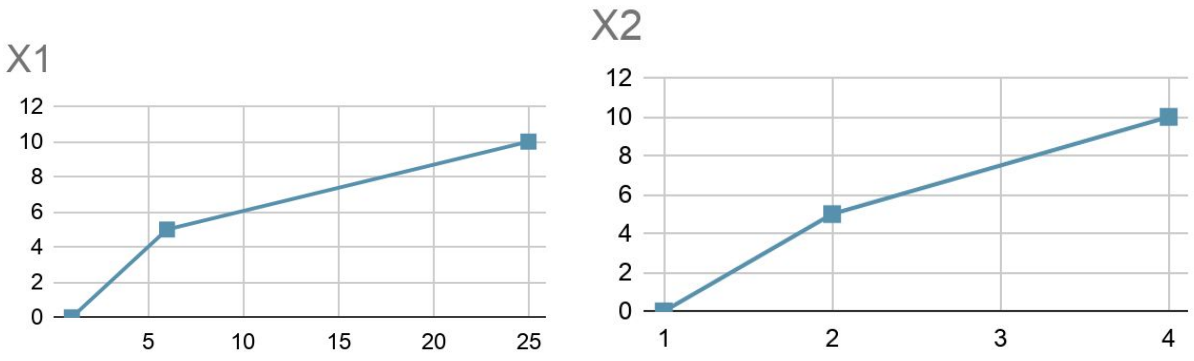


Рисунок 4.2 – X1, кількість платформ яку підтримує рушій; X2, швидкодія бази даних

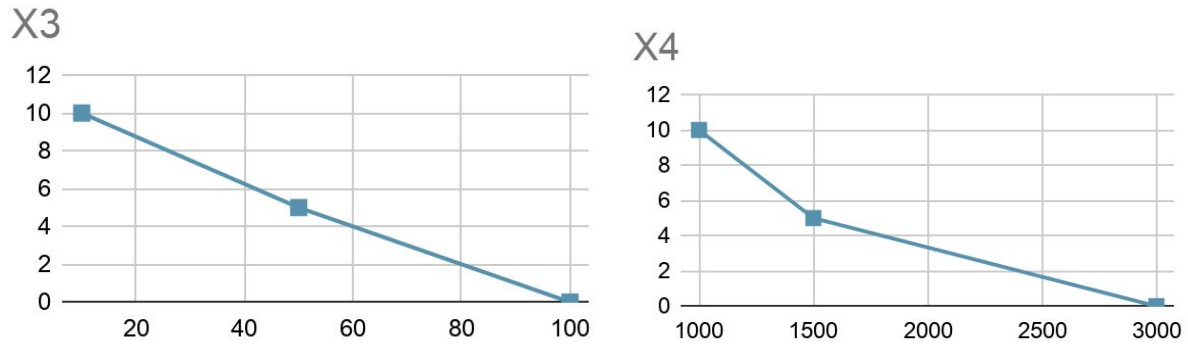


Рисунок 4.3 – X3, обробки запитів користувача; X4, потенційний об’єм програмного коду

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості всіх параметрів для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень. Значимість кожного параметра визначається методом попарного порівняння [26]. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значущості передбачає:

1) визначення рівня значимості параметра шляхом присвоєння різних рангів;

2) перевірку придатності експертних оцінок для подальшого використання;

3) визначення оцінки попарного пріоритету параметрів;

4) обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позн-ня пар-ра	Назва параметра	Од. виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δi	Δ_i^2
			1	2	3	4	5	6	7			
X1	Кількість платформ яку підтримує рушій	Оп/мс	1	2	2	1	2	2	3	13	-4.5	20.25
X2	Швидкодія бази даних	Мб	2	1	2	3	2	3	2	15	-2.5	6.25
X3	Час обробки запитів користувача	Мс	2	2	1	2	2	1	1	11	-6.5	42.25
X4	Потенційний об’єм програмного коду	кі-сть строк коду	5	5	5	4	4	4	4	31	13.5	182.25
	Разом		10	10	10	10	10	10	10	70	0	251

Для перевірки ступенядостовірності експертних оцінок, визначимо наступні параметри [26]:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70 \quad (1)$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5 \quad (2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T \quad (3)$$

де Т – середня сума рангів, R_i – сума рангів кожного параметра

Сума відхилень по всім параметрам повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 251 \quad (4)$$

де Δ_i^2 – відхилення, N – кількість експертів

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 251}{7^2(4^3-4)} = 1,02 > W_k = 0,67 \quad (5)$$

де S – загальна сума квадратів відхилення, W_k – нормативний коефіцієнт узгодженості.

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4. 4.

Таблиця 4. 4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	>	=	<	=	<	>	<	0,5
X1 і X3	<	=	>	<	=	>	>	>	1,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	=	<	>	>	=	>	>	>	1,5
X2 і X4	<	<	<	<	<	<	<	<	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*-го параметра над *j*-тим, a_{ij} визначається по формулі [26]:

$$a_{ij} = \{1,5 \text{ при } X_i > X_j \; 1,0 \text{ при } X_i = X_j \; 0,5 \text{ при } X_i < X_j \tag{6}$$

З отриманих числових оцінок переваги складемо матрицю A=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{\text{в}i} = \frac{b_i}{\sum\limits_{i=1}^{n} b_i}, \text{ де } b_i = \sum_{i=1}^{N} a_{ij} \tag{7}$$

де $K_{ві}$ – коефіцієнт вагомости *i*-го параметра

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{вi} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{ де } b_i' = \sum_{i=1}^{N} a_{ij} \qquad (8)$$

Як видно з таблиці 4. 5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4. 5 – Розрахунок вагомості параметрів

Параметри X_i	Параметри X_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	*bi*	*Кві*	*bi 1*	*Кві 1*	*bi 2*	*Кві 2*
X1	1	0.5	1.5	0.5	3.5	0.233	11.75	0.2259	40.625	0.2260
X2	1.5	1	1.5	0.5	4.5	0.3	15.75	0.3028	54.375	0.3025
X3	0.5	0.5	1	0.5	2.5	0.166	8.75	0.1682	30.375	0.1689
X4	1.5	0.5	1.5	1	4.5	0.3	15.75	0.3028	54.375	0.3025
Всього:					15	1	52	1	179.75	1

- 4.3 Аналіз рівня якості варіантів реалізації функцій

На основі порівняльного аналізу варіантів реалізації функцій за їх перевагами та недоліками і коефіцієнтами вагомості параметрів визначаємо рівень якості кожного варіанта виконання основних функцій окремо.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховують так [26]:

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} \quad (9)$$

де n – кількість параметрів;

$K_{вi}$ – коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Результати розрахунків зведено в таблиці 4.6

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	a	X1	1	1	0.23	0.23
	b	X1	6	6	0.23	1.13
F2	a	X3	50	4.5	0.17	0.76
F3	a	X2	40	4,5	0.3	1.36
	a	X4	1200	8	0.3	2.42

За даними з таблиці 4.6 за формулою

$$K_K = K_{\text{ТУ}}\left[F_{1k}\right] + K_{\text{ТУ}}\left[F_{2k}\right] + \cdots + K_{\text{ТУ}}\left[F_{zk}\right] \quad (10)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 0.23 + 0.76 + 1.36 + 2.42 = 4.77$$

$$K_{K2} = 1.13 + 0.76 + 1.36 + 2.42 = 5.67$$

Як видно з розрахунків, кращим є другий варіант, для якого коефіцієнт рівня якості має найбільше значення.

- 4.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка логічної частини програмного продукту;

2. Розробка візуальної частини програмного продукту.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3 [26].

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М} \quad (11)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру ступеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_Р$ = 90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою, загальна трудомісткість програмування першого завдання дорівнює:

$$Т_1 = 90*1.7*0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ = 27 людино-днів, $К_П$ = 0.9, $К_{СК}$ = 1, $К_{СТ}$ = 0.8:

$$Т_2 = 27*0.9*0.8 = 19.44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$Т_I = (122.4 + 19.44 + 4.8 + 19.44) * 8 = 1328.64 \text{ людино-годин;}$$
$$Т_{II} = (122.4 + 19.44 + 6.91 + 19.44) * 8 = 1345.52 \text{ людино-годин;}$$

Найбільш високу трудомісткість має варіант II.

В розробці бере участь 1 спеціаліст з мобільної розробки з окладом 14 160 грн та 1 спеціаліст, що займається написанням серверної частини та роботою з GCP з окладом у 14 160 грн. Визначимо зарплату за годину за формулою:

$$C_ч = \frac{M}{T_m \bullet t} \text{ грн.} \quad (12)$$

де M – місячний оклад працівників; T_m – кількість робочих днів тиждень; t – кількість робочих годин в день.

$$C_ч = \frac{2*14160}{2 \cdot 21 \cdot 8} = 84,28$$

Тоді, розрахуємо заробітну плату за формулою

$$C_{ЗП} = C_ч \cdot T_i \cdot K_Д \quad (13)$$

де $C_ч$ – величина погодинної оплати праці програміста; T_i – трудомісткість відповідного завдання; $K_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробника за варіантами становить:

I. $C_{ЗП} = 84,28 \cdot 1328,64 \cdot 1.2 = 134373.3$ грн.

II. $C_{ЗП} = 128,97 \cdot 1345.52 \cdot 1.2 = 136080.5$ грн.

Відрахування на єдиний соціальний внесок становить 22 %

Отримуємо такі значення:

I. $C_{ВІД} = C_{ЗП} \cdot 22 = 134373.3 * 22 = 29562.12$ грн.

II. $C_{ВІД} = C_{ЗП} \cdot 22 = 136080.5 * 22 = 29937.71$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($C_М$)

Так як одна ЕОМ обслуговує одного розробника з заробітною платою 14 160 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 14160 \cdot 0,2 = 33984 \text{ грн.} \quad (14)$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 33984 * (1 + 0.2) = 40780.8 \text{ грн.} \quad (15)$$

Відрахування на єдиний соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0.22 = 40780.8 * 0.22 = 8971.77 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 20 000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 * 0.25 * 20\,000 = 5750 \text{ грн.} \quad (16)$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 20\,000 \cdot 0.05 = 1150 \text{ грн.} \quad (17)$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 105 – 11 – 16) \cdot 8 \cdot 0.9 = 1677.6 \text{ год} \quad (18)$$

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування;

t–кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1677.6 * 0.22 * 0.78 * 2.4834 = 714{,}91 \text{ грн.} \quad (19)$$

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф 2,4834 грн за 1 КВт-годин електроенергії. Для прикладу ми підключилися як малі непобутові споживачі, що приєднані до мереж ПрАТ "ДТЕК Київські електромережі".

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 20\,000 * 0.67 = 13\,400 \text{ грн.} \quad (20)$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н \quad (21)$$

$$С_{ЕКС} = 40780.8 + 8971.77 + 5750 + 1150 + 714.91 + 13400 = 70767.48 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 70767.48 / 1706.4 = 41.47 \text{ грн/год.} \quad (22)$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М\text{-}Г} \cdot Т \quad (23)$$

I. $C_М = 41.47 * 1328.64 = 55098$ грн

II. $C_М = 41.47 * 1345.52 = 55798$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_Н = C_{ЗП} \cdot 0,67 \quad (24)$$

I. $C_Н = 134373.3 * 0.67 = 90030.11$ грн.;

II. $C_Н = 136080.5 * 0.67 = 91173.94$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н \quad (25)$$

I. $C_{ПП} = 134373.3 + 29562.12 + 55098 + 90030.11 = 309063$ грн.;

II. $C_{ПП} = 136080.5 + 29937.71 + 55798 + 91173.94 = 312990$ грн.;

4.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{ТЕР_j} = \frac{K_{Kj}}{C_{Фj}} \quad (26)$$

$К_{ТЕР1} = 5.68/ 309063 = 1.8 \cdot 10^{-5}$

$К_{ТЕР2} = 5.92/ 312990 = 1.9 \cdot 10^{-5}$;

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР2} = 1.9 \cdot 10^{-5}$

4.6 Висновки

В даному розділі проведено повний функціонально-вартісний аналіз створення програмного продукту, який було розроблено в рамках дипломного проекту. Процес аналізу було розділено на технічну та економічну частини.

Після виконання функціонально-вартісного аналізу програмного комплексу що розробляється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 1,6 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту був створений на графічному рушії Unity з використанням одразу 6 платформ і повністю відповідає заданим вимогам.

ВИСНОВКИ

У дипломній роботі досліджено функціональні можливості розповсюджених мобільних месенджерів, а саме проведено порівняльний аналіз трьох найпоширеніших сучасних мобільних додатків, які використовуються в Україні для спілкування - Whatsapp, Telegram та Rakuten Viber. Описані їх сильні та слабкі сторони з точку зору функціоналу та користувацьких можливостей.

Проаналізовано інструменти для створення клієнтської частини кросплатформенних мобільних додатків та проведено аналіз ринку функціонування хмарних серверних рішень. Зроблено порівняльний аналіз реляційних та нереляційних баз даних. Обрано графічний рушій Unity, Cloud платформу GCP, сервісне рішення App Engine, серверна мова програмування Python з фреймворком Flask, та реляційну базу даних PostgreSQL.

Створене архітектурне та дизайнерське рішення для мобільного месенджера. Описана реалізація клієнтської частини, написано API для клієнт - серверної взаємодії та створено базу даних.

У результаті проведеного дослідження в рамках даної роботи було створено протопип кросплатформеного чату с інноваційною гілкоподібної структурою для подальшого тестування зручності у використанні. Подібну структуру не має жоден з описаних існуючих мобільних месенджерів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційний сайт компанії KOMMANDOTECH. [Електронний ресурс] – Режим доступу: https://kommandotech.com/statistics/how-much-time-does-the-average-person-spend-on-their-phone/ . – Дата доступу : 09.10.2020.
2. Meucci, Sandra. Antonio and the Electric Scream: The Man Who Invented the Telephone – Boston: Branden Books, 2010 – 138p.
3. First Transatlantic Telephone Call Editors / History com. [Електроний ресурс] Режим доступу: https://www.history.com/topics/inventions/first-transatlantic-telephone-call-video . – Дата доступу : 09.10.2020.
4. Who Invented the Cell Phone? [Електроний ресурс] Режим доступу: https://science.howstuffworks.com/innovation/inventions/who-invented-the-cell-phone.htm . – Дата доступу : 09.10.2020.
5. Застосування мобільних додатків в бізнесі та їх облік [Електроний ресурс] / Т. М. Корпанюк, Я. І.Мулик//Електронне наукове фахове видання “Ефективна економіка”– 2018.– №3.– Режим доступу: http://www.economy.nayka.com.ua/pdf/3_2018/59.pdf. – Дата доступу : 15.11.2020.
6. Офіційний сайт компанії METRIXLAB. [Електронний ресурс] – Режим доступу: https://www.metrixlab.com/portfolio/messaging-apps-change-way-communicate/ . – Дата доступу : 09.10.2020.
7. Офіційний сайт компанії STATISTA. [Електронний ресурс] – Режим доступу: https://www.statista.com/statistics/195140/new-user-generated-content-uploaded-by-users-per-minute/ . – Дата доступу : 09.10.2020.
8. Офіційний сайт компанії WHATSAPP [Електронний ресурс] – Режим доступу: https://www.whatsapp.com/features/ . – Дата доступу : 09.10.2020.

9. Офіційний сайт компанії SIGNAL [Електронний ресурс] – Режим доступу https://signal.org/en/. – Дата доступу : 09.10.2020.
10. Cohn-Gordon, Katriel. “A Formal Security Analysis of the Signal Messaging Protocol.” Cryptology, [Електронний ресурс] – Режим доступу: https://eprint.iacr.org/2016/1013.pdf . – Дата доступу : 09.10.2020.
11. Офіційний сайт компанії TELEGRAM [Електронний ресурс] – Режим доступу: https://telegram.org/faq?setln=uk . – Дата доступу : 09.10.2020.
12. Google store [Електронний ресурс] – Режим доступу: https://play.google.com/store/apps/details?id=org.telegram.messenger . – Дата доступу : 09.10.2020.
13. Офіційний сайт компанії RAKUTEN VIBER [Електронний ресурс] – Режим доступу: https://www.viber.com/en/ . – Дата доступу : 09.10.2020.
14. Офіційний сайт компанії SIMILARWEB [Електронний ресурс] – Режим доступу: https://www.similarweb.com/app/google-play/com.viber.voip/statistics/#ranking . – Дата доступу : 09.10.2020.
15. Cross Platform Mobile App Development Guide [Електронний ресурс] – Режим доступу: https://www.businessofapps.com/guide/cross-platform-mobile-app-development/. – Дата доступу : 09.10.2020.
16. Flutter vs Xamarin vs React Native - Which cross platform Mobile App Development framework to choose in 2020 [Електронний ресурс] – Режим доступу: https://habr.com/ru/sandbox/138374/. – Дата доступу : 09.10.2020.
17. Офіційний сайт компанії UNITY [Електронний ресурс] – Режим доступу: https://unity.com. – Дата доступу : 09.10.2020.
18. Офіційний блог компанії UNITY [Електронний ресурс] – Режим доступу: https://blogs.unity3d.com/2017/08/11/unityscripts-long-ride-off-into-the-sunset/. – Дата доступу : 09.10.2020.

19. Офіційний сайт компанії AMAZON WEB SERVICES [Електронний ресурс] – Режим доступу: https://aws.amazon.com/ru/?nc2=h_lg. – Дата доступу : 09.10.2020.
20. AWS vs. Azure vs. Google: Detailed Cloud Comparison [Електронний ресурс] – Режим доступу: https://levelup.gitconnected.com/aws-vs-azure-vs-google-detailed-cloud-comparison-b075a35fc8b8. – Дата доступу : 09.10.2020.
21. Офіційний сайт компанії MICROSOFT AZURE [Електронний ресурс] – Режим доступу: https://azure.microsoft.com/en-us/. – Дата доступу : 09.10.2020.
22. Офіційний сайт компанії GOOGLE CLOUD PLATFORM [Електронний ресурс] – Режим доступу: https://cloud.google.com/. – Дата доступу : 09.10.2020.
23. Choosing Java vs Python on Google App Engine [Електронний ресурс] – Режим доступу: https://stackoverflow.com/questions/1085898/choosing-java-vs-python-on-google-app-engine. – Дата доступу : 09.10.2020.
24. Електронний посібник для девелоперів від компанії GOOGLE [Електронний ресурс] – Режим доступу: https://developers.google.com/learn/topics/python. – Дата доступу : 09.10.2020.
25. Електронний посібник для девелоперів від компанії MOZILLA [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Glossary/MVC. – Дата доступу : 09.10.2020.
26. Пашін В.П. Методичні вказівки до виконання економічно-організаційного розділу дипломних проектів (робіт) бакалаврів та спеціалістів для студентів інституту прикладного системного аналізу/ В.П. Пашін, В.В. Романов, Н.В. Єгорова.// НТУУ “КПІ”. – 2011. – С.118.

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
using System;
using UnityEngine;

[Serializable]
public class Chat
{
    public string parentChatId;
    public string chatId;
    public string chatName;
    public string[] chatSideIds;
    public long bottomChatId;
    public long lastMessageSeen;
}



using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ChatListViewer : MonoBehaviour
{
    public ScrollRect scrollRect;
    public GameObject content;
    public GameObject channelViewPrefab;

    private void Awake()
    {
        //TODO Clear chat on awake
        BuildChatsView(ChatsHolder.Instance.chats.ToArray());
    }

    public void BuildChatsView(Chat[] chats)
    {
        RectTransform contentObj = ((RectTransform)content.transform);
        //Find root chats
        List<Chat> rootChats = new List<Chat>();
        foreach (Chat chat in chats) if (chat.parentChatId == null || chat.parentChatId == "")
AddChatToList(chat);
    }

    public void AddChatToList(Chat chat)
```

```
    {
        GameObject chatInstance = Instantiate(channelViewPrefab);
        chatInstance.GetComponent<ChatViewController>().InitChatView(chat);
        chatInstance.SetActive(false);
        chatInstance.transform.SetParent(content.transform);
        SetRectTransformProperties((RectTransform)channelViewPrefab.transform,
(RectTransform)chatInstance.transform);
        StartCoroutine(MessageViewBugHack(chatInstance));
    }

    IEnumerator MessageViewBugHack(GameObject messageObj)
    {
        yield return new WaitForFixedUpdate();
        messageObj.SetActive(true);
    }

    private void SetRectTransformProperties(RectTransform from, RectTransform to)
    {
        to.sizeDelta = from.sizeDelta;
        to.localScale = from.localScale;
    }
}



using System;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

public class ChatsHolder : MonoBehaviour
{
    public static ChatsHolder Instance
    {
        get
        {
            if (m_Instance == null)
            {
                //Find singleton
                m_Instance = (ChatsHolder)FindObjectOfType(typeof(ChatsHolder));
                // Create new instance if one doesn't already exist.
                if (m_Instance == null)
                {
                    // Need to create a new GameObject to attach the singleton to.
                    var singletonObject = new GameObject();
                    m_Instance = singletonObject.AddComponent<ChatsHolder>();
                    singletonObject.name = "ChatsHolder (Singleton)";
```

```
            }
        }

        return m_Instance;
    }
}
private static ChatsHolder m_Instance;

public static string URL;
public List<Message> messages = new List<Message>();
public long lastMessageTimestamp;
public List<Chat> chats = new List<Chat>();
public UnityEvent messagesUpdated = new UnityEvent();
public UnityEvent chatsUpdated = new UnityEvent();

public void SetAllMessages(Message[] allMessages)
{
    messages = new List<Message>(allMessages);
    messages.Sort(new Comparison<Message>((x, y) => (int)(x.timestamp - y.timestamp)));
}

public void UpdateMessagesReceived(Message[] newMessages)
{
    //filter out repeated
    foreach (Message newMessage in newMessages)
    {
        if (FindWithSameId(messages, newMessage) == null) messages.Add(newMessage);
    }
    messages.Sort(new Comparison<Message>((x, y) => (int)(x.timestamp - y.timestamp)));
}

public void UpdateChatsReceived(Chat[] updatedChats)
{
    List<Chat> newChats = new List<Chat>();
    foreach (Chat chat in updatedChats)
    {
        if (FindChatById(chat.chatId) == null) chats.Add(chat);
        else FindChatById(chat.chatId).chatName = chat.chatName;
    }
}

public Message[] GetReadMessagesByChat(Chat chat)
{
    List<Message> filtered = new List<Message>();
    foreach (Message message in messages)
    {
        if (message.chatId == chat.chatId && IsMessageViewed(message)) filtered.Add(message);
```

```
    }

    return filtered.ToArray();
}

public Message[] GetUnreadMessagesByChat(Chat chat)
{
    List<Message> filtered = new List<Message>();
    foreach (Message message in messages)
    {
        if (message.chatId == chat.chatId && !IsMessageViewed(message)) filtered.Add(message);
    }

    return filtered.ToArray();
}

public Chat[] GetChildChats(Chat parentChat)
{
    List<Chat> result = new List<Chat>();
    foreach (Chat chat in chats)
    {
        if (chat.parentChatId == parentChat.chatId) result.Add(chat);
    }

    return result.ToArray();
}

public Chat FindChatById(string chatId)
{
    foreach (Chat chat in chats) if (chat.chatId == chatId) return chat;

    return null;
}

private Message FindWithSameId(List<Message> filter, Message value)
{
    foreach (Message message in filter) if (message.messageId == value.messageId) return value;

    return null;
}

private bool IsMessageViewed(Message message)
{
    if (message.senderId == Statics.UserId) return true;

    return message.messageviewed[0];
}
```

```
}



using UnityEngine;
using UnityEngine.UI;

public class ChatViewController : MonoBehaviour
{
    public Text chatId;
    public ChatScreenLoader chatScreenLoader;
    [HideInInspector]
    public Chat chat;

    public void InitChatView(Chat chatValue)
    {
        //set login for chat common window and chatname for chat with a parent
        chat = chatValue;
        chatId.text = (chat.parentChatId == null || chat.parentChatId == "") ?
GetNotEqualString(chatValue.chatSideIds, Statics.UserId) : chat.chatName;
    }

    public void OpenChat()
    {
        chatScreenLoader.currentChat = chat;
        UIStateMachine.Instance.ChangeState(UIStateMachine.UIState.OneChatView);
    }

    public void RefreshChatScreen()
    {
        chatScreenLoader.RefreshChatScreen(chat);
    }
    private string GetNotEqualString(string[] arr, string compar)
    {
        foreach (string str in arr) if (str != compar) return str;

        return "";
    }
}



using UnityEngine;
using UnityEngine.Events;
using UnityEngine.UI;

public class LoginBtnController : MonoBehaviour
```

```
{
    public InputField login;
    public InputField passw;

    public void Login()
    {
        UnityEvent callback = new UnityEvent();
        callback.AddListener(LoginIsOk);
        NetworkingManager.Instance.Login(login.text, passw.text, callback);
    }

    private void LoginIsOk()
    {
        UIStateMachine.Instance.ChangeState(UIStateMachine.UIState.ChatsView);
    }
}
```

```
using System;

[Serializable]
public class Message
{
    public string senderId;
    public string[] sideIds;
    public string chatId;
    public string messageId;
    public long timestamp;
    public string messageText;
    public bool[] messageviewed;
}
```

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ChatNameController : MonoBehaviour
{
    public Text chateName;

    public void InitChatHeader(Chat chat)
    {
        chateName.text = chat.chatName;
    }
```

```
}



using UnityEngine;

public class MessageIsViewedController : MonoBehaviour
{
    public RectTransform contentObj;
    private Message message;

    private void OnEnable()
    {
        message = GetComponent<MessageViewController>().message;
    }

    private void Update()
    {
        if (message.senderId == Statics.UserId) return;
        if (message.messageviewed[0]) return;

        //check for canvas
        Vector3[] vectors = new Vector3[4];
        //contentObj.GetWorldCorners(vectors);
        //float minContentHeight = MinimalYValue(vectors);
        ((RectTransform)transform).GetWorldCorners(vectors);
        float minMessageHeight = MinimalYValue(vectors);
        //message.messageviewed[0] = minContentHeight < minMessageHeight + 20;
        message.messageviewed[0] = minMessageHeight > 100;
        if (message.messageviewed[0]) NetworkingManager.Instance.UpdateMessage(message);
    }

    private float MinimalYValue(Vector3[] arr)
    {
        float[] ys = { arr[0].y, arr[1].y, arr[2].y, arr[3].y };
        return Mathf.Min(ys);
    }
}



using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
```

```
public class MessageViewBuilder : MonoBehaviour
{
    public ScrollRect scrollRect;
    public GameObject content;
    public GameObject incomingMessagePrefab;
    public GameObject outgoingMessagePrefab;
    [HideInInspector]
    public List<Message> builtMessages = new List<Message>();

    public void BuildMessagesChatView(Message[] messages, Message[] unreadMessages)
    {
        //clear all prev messages
        RectTransform contentObj = ((RectTransform)content.transform);
        int prevMessagesNumber = contentObj.childCount;
        for (int i = prevMessagesNumber - 1; i > 1; i--) {
            contentObj.GetChild(i).gameObject.SetActive(false);
            Destroy(contentObj.GetChild(i).gameObject);
        }
        builtMessages.Clear();
        //add read messages
        List<Message> messagesList = new List<Message>(messages);
        foreach (Message message in messagesList) AddMessageToChat(message);
        StartCoroutine(ScrollDownDelayed(unreadMessages));
    }

    IEnumerator ScrollDownDelayed(Message[] unreadMessages)
    {
        yield return new WaitForSeconds(0.2F);
        scrollRect.verticalNormalizedPosition = 0F;
        foreach (Message message in unreadMessages) AddMessageToChat(message);
        yield return new WaitForSeconds(0.1F);
        //hack
        GameObject mimic = GameObject.Instantiate(incomingMessagePrefab, content.transform);
        yield return new WaitForSeconds(0.1F);
        Destroy(mimic);
    }

    IEnumerator ScrollDownDelayed()
    {
        yield return new WaitForSeconds(0.2F);
        scrollRect.verticalNormalizedPosition = 0F;
    }

    public void AddNewMessages(Message[] messages)
    {
        foreach (Message message in messages) AddMessageToChat(message);
    }
```

```
    public void AddMessageTyped(Message message)
    {
        AddMessageToChat(message);
        StartCoroutine(ScrollDownDelayed());
    }

    public void AddMessageToChat(Message message)
    {
        GameObject prefab = Statics.UserId == message.senderId ? outgoingMessagePrefab :
incomingMessagePrefab;
        GameObject messageInstance = Instantiate(prefab);
        messageInstance.GetComponent<MessageViewController>().InitView(message);
        messageInstance.SetActive(true);
        messageInstance.transform.SetParent(content.transform);
        SetRectTransformProperties((RectTransform)prefab.transform,
(RectTransform)messageInstance.transform);
        builtMessages.Add(message);
    }

    private void SetRectTransformProperties(RectTransform from, RectTransform to)
    {
        to.sizeDelta = from.sizeDelta;
        to.localScale = from.localScale;
    }
}



using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

public class MessageViewController : MonoBehaviour , IPointerDownHandler , IPointerUpHandler
{
    public Text text;
    public GameObject branchBtn;
    [HideInInspector]
    public Message message;
    private float clickStart;
    private static float SHOW_BRANCH_BUTTON_TIMER = 2F;
    private bool mouseIsPressed;

    public void InitView(Message message)
    {
        this.message = message;
        text.text = message.messageText;
```

```
}

void Update()
{
    if (!mouseIsPressed)
    {
        clickStart = 0;

        return;
    }
    if (mouseIsPressed)
    {
        clickStart += Time.deltaTime;
    }

    if (clickStart > SHOW_BRANCH_BUTTON_TIMER)
    {
        SwitchBranchButton();
        clickStart = 0;
    }
}

private void SwitchBranchButton()
{
    branchBtn.SetActive(!branchBtn.activeSelf);
}

public void OnPointerDown(PointerEventData eventData)
{
    mouseIsPressed = true;
}

public void OnPointerUp(PointerEventData eventData)
{
    mouseIsPressed = false;
}

public void BranchPressed()
{
    SwitchBranchButton();
    //create new chat with current chat as parent
    Chat currentChat = ChatsHolder.Instance.FindChatById(message.chatId);
    Chat newChat = new Chat()
    {
        parentChatId = currentChat.chatId,
        chatId = System.DateTime.Now.Ticks.ToString(),
        chatName = "New Branch Chat",
```

```
            chatSideIds = currentChat.chatSideIds,
            bottomChatId = 1L,
            lastMessageSeen = 1L
        };
        //add Chat to Holder
        ChatsHolder.Instance.UpdateChatsReceived(new Chat[] { newChat });
        //push chat to server
        NetworkingManager.Instance.SendNewChat(newChat);
        //create duplicate message with new chat as parent
        Message newMessage = new Message() {
            senderId = message.senderId,
            sideIds = message.sideIds,
            chatId = newChat.chatId,
            messageId = System.DateTime.Now.Ticks.ToString(),
            timestamp = 1L,
            messageText = message.messageText,
            messageviewed = new bool[] {true, true}
        };
        //push new message to server
        NetworkingManager.Instance.SendMessage(newMessage);
        //add message to ChatHolder
        ChatsHolder.Instance.UpdateMessagesReceived(new Message[] { newMessage });
    }
}



using UnityEngine;

public class NewMessagesIndicatorController : MonoBehaviour
{
    public MessageViewBuilder messageViewBuilder;
    public GameObject graphics;

    void Update()
    {
        if (!messageViewBuilder.gameObject.activeInHierarchy)
        {
            graphics.SetActive(false);
            return;
        }

        Message[] messages = messageViewBuilder.builtMessages.ToArray();
        int counter = 0;
        foreach (Message message in messages)
        {
            if (message.senderId == Statics.UserId) continue;
```

```
            if (!message.messageviewed[0]) counter++;
        }
        graphics.SetActive(counter > 0);
    }
}



using System;
using System.Collections.Generic;
using UnityEngine;

public class OneChatVerticalView : MonoBehaviour
{
    private static float MESSAGES_VIEW_HEIGHT = 1400;
    private static float MESSAGES_VIEW_WIDTH = 800;

    public ChatNameController chatNameController;
    public RectTransform verticalView;
    public RectTransform messagesView;
    public GameObject chatViewPrefab;
    [HideInInspector]
    public List<GameObject> chatViews = new List<GameObject>();

    public void InitChatViews(Chat rootChat)
    {
        //clearing existing chats buttons
        foreach (GameObject oldChatView in chatViews.ToArray()) Destroy(oldChatView);
        chatViews.Clear();
        //getting chats to show
        Chat[] childChats = ChatsHolder.Instance.GetChildChats(rootChat);
        //if zero chats, quit and dont build
        if (childChats.Length == 0)
        {
            UpdateMessagesViewHeight(0);
            return;
        }

        foreach (Chat chat in childChats) BuildChatViews(chat);
        UpdateMessagesViewHeight(childChats.Length);
        //switch message view to last position
        messagesView.SetParent(null);
        messagesView.SetParent(verticalView);
    }

    private void UpdateMessagesViewHeight(int count)
    {
```

```
        float prefabHeight = ((RectTransform)chatViewPrefab.transform).rect.height;
        messagesView.sizeDelta = new Vector2(MESSAGES_VIEW_WIDTH,
MESSAGES_VIEW_HEIGHT - prefabHeight * count);
    }

    private void BuildChatViews(Chat chat)
    {
        GameObject prefabInstance = Instantiate(chatViewPrefab, verticalView);
        prefabInstance.GetComponent<ChatViewController>().InitChatView(chat);
        chatViews.Add(prefabInstance);
        prefabInstance.SetActive(true);
    }
}



using UnityEngine;
using UnityEngine.UI;

public class SendMessageController : MonoBehaviour
{
    public InputField inputField;
    public ChatScreenLoader chatScreenLoader;
    public ScrollRect scrollRect;
    public void SendMessage()
    {
        Message message = new Message()
        {
            senderId = Statics.UserId,
            sideIds = new string[] { Statics.UserId, chatScreenLoader.GetSideChatId() },
            chatId = chatScreenLoader.currentChat.chatId,
            messageId = Random.Range(0, 100000).ToString(),
            timestamp = 1L,
            messageviewed = new bool[] { true, false }
        };
        inputField.text = "";
        chatScreenLoader.NewMessageTyped(message);

    }
}



using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
public class ChatScreenLoader : MonoBehaviour
{
    public GameObject placeHolder;
    public ChatNameController chatNameController;
    public MessageViewBuilder messageViewBuilder;
    public OneChatVerticalView oneChatVerticalView;
    [HideInInspector]
    public Chat currentChat;
    [HideInInspector]
    public Stack<Chat> previousChats = new Stack<Chat>();
    private int messagesCount;

    private void OnEnable()
    {
        InitChat();
    }
    public void InitChat()
    {
        //cleaning stack of prev states
        previousChats.Clear();
        chatNameController.InitChatHeader(currentChat);

messageViewBuilder.BuildMessagesChatView(ChatsHolder.Instance.GetReadMessagesByChat(current
Chat), ChatsHolder.Instance.GetUnreadMessagesByChat(currentChat));
        messagesCount = ChatsHolder.Instance.GetReadMessagesByChat(currentChat).Length +
ChatsHolder.Instance.GetUnreadMessagesByChat(currentChat).Length;
        oneChatVerticalView.InitChatViews(currentChat);
        SetListeners();
    }

    private void SetListeners()
    {
        StartCoroutine(UpdateChatScreen());
    }

    private void OnDisable()
    {
        StopAllCoroutines();
    }

    IEnumerator UpdateChatScreen()
    {
        while (true)
        {
            yield return new WaitForSeconds(1F);

            chatNameController.InitChatHeader(ChatsHolder.Instance.FindChatById(currentChat.chatId));
```

```
        //checking if any new messages for current chat arrived
        int newMessagesCount = ChatsHolder.Instance.GetReadMessagesByChat(currentChat).Length
+ ChatsHolder.Instance.GetUnreadMessagesByChat(currentChat).Length;
        if (messagesCount != newMessagesCount)
        {
            List<Message> newMessages = new List<Message>();

newMessages.AddRange(ClearIntersectionMessages(messageViewBuilder.builtMessages.ToArray(),
ChatsHolder.Instance.GetReadMessagesByChat(currentChat)));

newMessages.AddRange(ClearIntersectionMessages(messageViewBuilder.builtMessages.ToArray(),
ChatsHolder.Instance.GetUnreadMessagesByChat(currentChat)));
            if (newMessages.Count > 0)
            {
                messageViewBuilder.AddNewMessages(newMessages.ToArray());
            }
        }

        // check for branch chats number, if it is different - redraw chat
        int newCount = ChatsHolder.Instance.GetChildChats(currentChat).Length;
        int oldCount = oneChatVerticalView.chatViews.Count;
        if (newCount != oldCount)
        {
            oneChatVerticalView.InitChatViews(currentChat);
        }
    }
}

public void NewMessageTyped(Message message)
{
    NetworkingManager.Instance.SendMessage(message);
    ChatsHolder.Instance.UpdateMessagesReceived(new Message[] { message });
    messagesCount++;
    //add and scroll to the end of list
    messageViewBuilder.AddMessageTyped(message);
}

public void RefreshChatScreen(Chat newChat)
{
    previousChats.Push(currentChat);
    currentChat = newChat;
    chatNameController.InitChatHeader(newChat);

messageViewBuilder.BuildMessagesChatView(ChatsHolder.Instance.GetReadMessagesByChat(newCh
at), ChatsHolder.Instance.GetUnreadMessagesByChat(currentChat));
```

```
        messagesCount = ChatsHolder.Instance.GetReadMessagesByChat(currentChat).Length +
ChatsHolder.Instance.GetUnreadMessagesByChat(currentChat).Length;
        oneChatVerticalView.InitChatViews(newChat);
    }

    public void BackChat()
    {
        if (previousChats.Count == 0)
        {
            HomeChat();
            return;
        }

        currentChat = previousChats.Pop();
        chatNameController.InitChatHeader(currentChat);

messageViewBuilder.BuildMessagesChatView(ChatsHolder.Instance.GetReadMessagesByChat(current
Chat), ChatsHolder.Instance.GetUnreadMessagesByChat(currentChat));
        oneChatVerticalView.InitChatViews(currentChat);
    }

    public void HomeChat()
    {
        UIStateMachine.Instance.ChangeState(UIStateMachine.UIState.ChatsView);
    }

    public string GetSideChatId()
    {
        if (currentChat.chatSideIds[0] == Statics.UserId) return currentChat.chatSideIds[1];

        return currentChat.chatSideIds[0];
    }

    private Message[] ClearIntersectionMessages(Message[] messages1, Message[] messages2)
    {
        List<Message> intersection = new List<Message>();
        foreach (Message message1 in messages1)
        {
            bool isFound = false;
            foreach (Message message2 in messages2)
            {
                if (message1.messageId == message2.messageId) isFound = true;
                break;
            }
            if (!isFound) intersection.Add(message1);
        }
        foreach (Message message2 in messages2)
```

```
        {
            bool isFound = false;
            foreach (Message message1 in messages1)
            {
                if (message2.messageId == message1.messageId) isFound = true;
                break;
            }
            if (!isFound) intersection.Add(message2);
        }

        return intersection.ToArray();
    }
}



using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Statics
{
#if UNITY_EDITOR
    public static string UserId = "TestId";
#else
    public static string UserId;
#endif
}



using UnityEngine;

public class UIStateMachine : MonoBehaviour
{
    public static UIStateMachine Instance
    {
        get
        {
            if (m_Instance == null)
            {
                //Find singleton
                m_Instance = (UIStateMachine)FindObjectOfType(typeof(UIStateMachine));
                // Create new instance if one doesn't already exist.
                if (m_Instance == null)
                {
                    // Need to create a new GameObject to attach the singleton to.
```

```
                var singletonObject = new GameObject();
                m_Instance = singletonObject.AddComponent<UIStateMachine>();
                singletonObject.name = "UIStateMachine (Singleton)";
            }
        }

        return m_Instance;
    }
}
private static UIStateMachine m_Instance;

public enum UIState { Login = 1, ChatsView = 2, OneChatView = 3, BranchView = 4 }
public GameObject loginView;
public GameObject chatsView;
public GameObject chatView;
public GameObject branchView;

public void ChangeState(UIState newState)
{
    loginView.SetActive(false);
    chatsView.SetActive(false);
    chatView.SetActive(false);
    branchView.SetActive(false);

    switch ((int) newState)
    {
        case ((int)UIState.Login):
            loginView.SetActive(true);
            break;
        case ((int)UIState.ChatsView):
            chatsView.SetActive(true);
            break;
        case ((int)UIState.OneChatView):
            chatView.SetActive(true);
            break;
        case ((int)UIState.BranchView):
            branchView.SetActive(true);
            break;
    }
}
}
```

ДОДАТОК Б ІЛЮСТРАТИВНИЙ МАТЕРІАЛ

Розповсюджені мобільні месенджери в Україні та світі

WhatsApp Viber Telegram

Опис ідеї

Звичайний лінійний чат

- Обговорення покупок в магазині
- Обговорення що подарувати на новий рік
- Обговорення новин по роботі
- Обговорення захисту диплома
- Обговорення побутових питань

Лінійний чат з каналами

Покупки	Text
Нов рік	Text
Новини	Text
Диплом	Text
Побут	Text
Навчання	

Чат з гілками

Технології

Клієнт

unity

REST API

Google Cloud Platform

App Engine

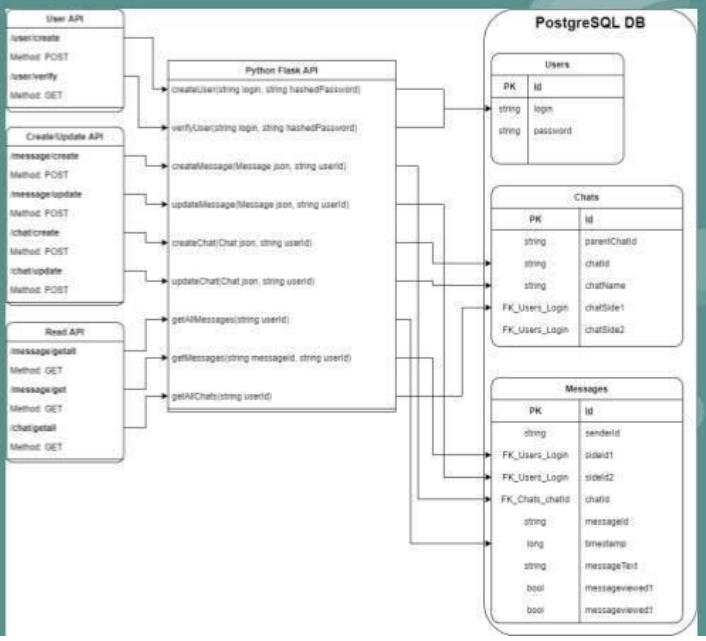
SQL

Flask

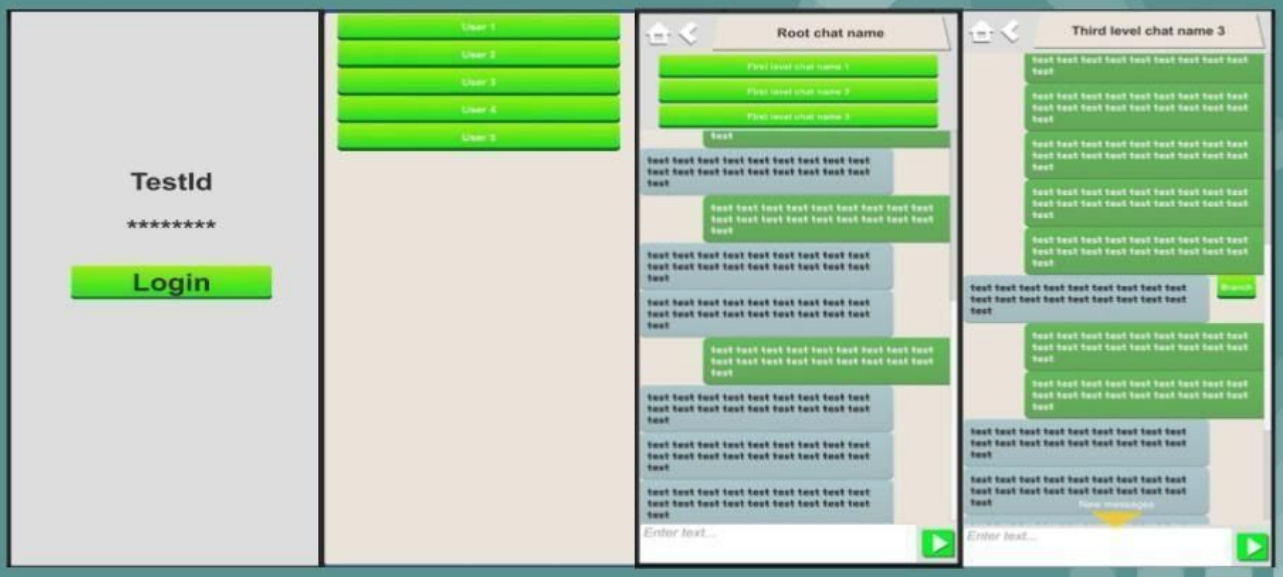
python

PostgreSQL

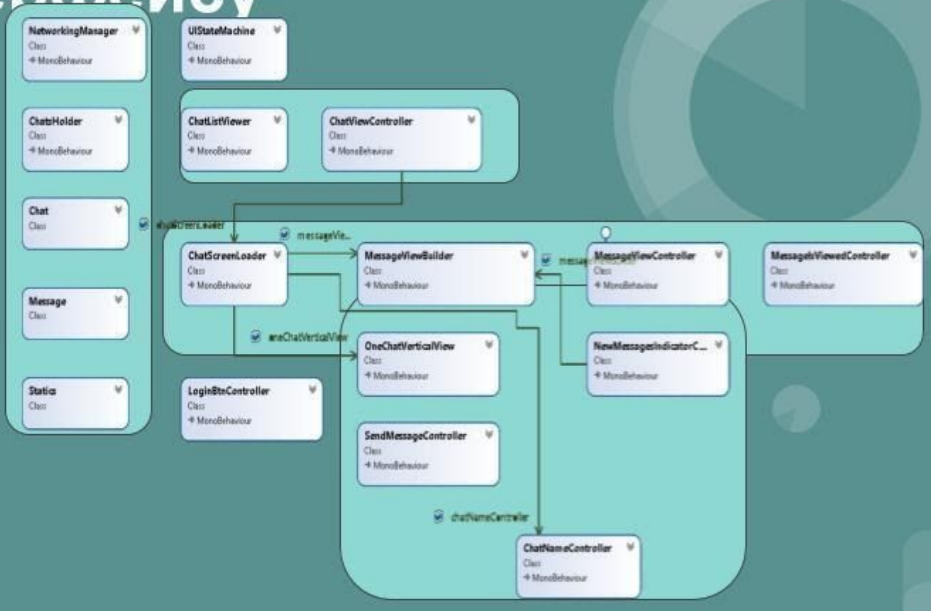
Діаграма бекенду та бази даних



Інтерфейс клієнта



Діаграма класів реалізації інтерфейсу



Висновки роботи

У дипломній роботі досліджено функціональні можливості розповсюджених мобільних месенджерів, а саме проведено порівняльний аналіз трьох найпоширеніших сучасних мобільних додатків, які використовуються в Україні для спілкування - Whatsapp, Telegram та Rakuten Viber. Описані їх сильні та слабкі сторони з точку зору функціоналу та користувацьких можливостей.

Проаналізовано інструменти для створення клієнтської частини кросплатформенних мобільних додатків та проведено аналіз ринку функціонування хмарних серверних рішень. Зроблено порівняльний аналіз реляційних та нереляційних баз даних. Обрано графічний рушій Unity, Cloud платформу GCP, сервісне рішення App Engine, серверна мова програмування Python з фреймворком Flask, та реляційну базу даних PostgreSQL.

Спроектовано архітектурне та дизайнерське рішення для мобільного месенджера. Описана реалізація клієнтської частини, написано API для клієнт - серверної взаємодії та створено базу даних.

У результаті проведеного дослідження в рамках даної роботи було створено протопип кросплатформеного чату с інноваційною гілкоподібної структурою для подальшого тестування зручності у використанні. Подібну структуру не має жоден з описаних існуючих мобільних месенджерів.