Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра мультимедійних систем

Курсова робота

освітній ступінь – бакалавр

на тему: «Р**озробка віртуального асистента для месенджеру**»

Виконав: студент 3-го року навчання,
Спеціальності
122 комп’ютерні науки
Макаренко Б. П.
Керівник Калітовський Б.В..,
_____
магістр комп’ютерних наук, асистент
«____» _____ 20____ р.

Київ – 2021

Київ-2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. Кафедри мультимедійних систем,
доцент, к.ф-м.н.
_____ О.П. Жежерун

(підпис)
“____” _____ 2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

Студенту Макаренко Богдану Петровичу факультету інформатики 3 курсу

ТЕМА Розробка віртуального асистента для месенджеру

Зміст ТЧ до курсової роботи:

Календарний план виконання курсової роботи

Вступ

Дослідження та теоретичний аналіз серверних технологій

Дослідження та теоретичний аналіз фронтенд технологій

Опис реалізації програмного продукту

Висновки

Список використаних джерел

Додатки

Дата видачі “____” _____ 2021 р. Керівник _____

(підпис)

Завдання отримав _____

Календарний план виконання курсової роботи

Тема: Розробка віртуального асистента для месенджеру

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Розгляд та аналіз проблеми	Вересень - грудень 2020р.	
2.	Розробка архітектури та програми аналіз технологій	Січень-лютий 2021р.	
3.	Розробка back-end частини	Лютий-березень 2021р.	
4.	Розробка front-end частини	Березень-квітень 2021р.	
5.	Написання текстової частини	Квітень 2021р.	
6.	Перегляд праці науковим керівником	Травень 2021р.	
7.	Підготовка до презентації роботи	Травень 2021р.	
8.	Презентація курсової роботи	Травень 2021р.	

Студент Макаренко Б.П _____

Керівник Калітовський Б.В. _____

“_____” _____ 2021 р.

ЗМІСТ

Анотація

Мета цієї роботи – створення безпечного, зручного, функціонального застосунку на мобільні пристрої для обміну текстовими повідомленнями. Особливістю месенджеру виступає вбудований віртуальний асистент.

Проведено аналіз існуючих додатків та визначено базовий функціонал месенджеру та віртуального асистенту. Досліджено сучасні технології та підходи до розробки як серверних так і клієнтських застосунків.

Після аналізу було розроблено два основних компоненти – сервер та клієнтський додаток.

Вступ

Завдяки технічним можливостям сучасні месенджери пропонують набагато більше функцій ніж простий обмін текстовими повідомленнями. Вони грають роль своєрідних компаньйонів які надають швидкий доступ до інформації. Через це вони набули широкого поширення в бізнесі, державних установах та серед пересічних користувачів.

Першим важливим етапом є дослідження предметної області, аналіз сучасних застосунків обміну повідомленнями, пошук аналогічних проектів. Проаналізувавши такі дані було визначено який функціонал є дійсно корисним та користується попитом у користувачів.

Наступним етапом є вибір технологій, дослідження їхньої документації та можливостей. Для реалізації було вирішено використовувати клієнт-серверну архітектуру. Значну увагу було приділено безпеці передачі даних між клієнтом та сервером, безпеці зберігання конфіденційної інформації у БД.

Останній етап – реалізація усіх частин проекту, їх тестування, публікація необхідних частин у хмарні сервіси.

Основні частини створені в результаті розробки:

- TCP сервер на C# .NET Core 5.0
- Мобільний застосунок з використанням технологій React Native, Redux, Java
- База даних для збереження інформації про користувачів, даних чатів.

Було використане таке програмне забезпечення:

- Середовища розробки: Visual Studio 2019, VS Code, Android Studio, IntelliJ IDEA 2020
- Мови програмування: Javascript, Java, C#
- Система керування версіями: Git
- Сервіс для розміщення серверу: Google Cloud Platform (а саме Compute Engine > VM instance)

- Операційні системи: Windows 10, Debian 10
- Бази даних: MongoDB
- Додаткове ПО для розробки/тестування: MongoDB Compass, Wireshark, Packet Sender.

Робота складається з вступу, чотирьох частин, висновків, списку використаних джерел та додатків.

Розділ 1. Аналіз предметної області. Постановка завдання курсової роботи

1.1 Проблематика та актуальність

Сучасні месенджери взяли на себе підтримку багатьох функцій(не тільки відправки або отримання повідомлень): вони можуть надавати користувачу інформацію з будь-яких сфер життя, дозволяють комунікувати з різними бізнес-організаціями, державними установами, планувати події і т.д. Зазвичай такі функції підтримують боти, які набули широкої популярності. Такі боти створюються сторонніми розробниками, а офіційні розробники месенджерів лише надають ним зручне API.

Мета цього проекту - створити месенджер який буде мати вбудований бот-асистент. Асистент надає функціонал який зазвичай надають окремі боти. Окрім того коли користувачу потрібно швидко зробити якусь дію, не дуже зручно шукати точну команду яка її виконує. Набагато зручніше коли асистент може сприймати природну текстову мову. Це підтверджують незліченна кількість віртуальних помічників як з голосовим так і тестовим розпізнаванням мови. Наприклад: «Google Assistant», «Аліса», «Siri».

1.2 Аналіз аналогічних продуктів, їх переваги та недоліки

Telegram – потужний сучасний месенджер. Зі слів розробників найбільш захищений серед усіх популярних аналогів. Функцій звичайного месенджера в ньому більш ніж достатньо – секретні, приватні, групові чати, публічні канали та можливість створення власних ботів. Для простого користувача тут доступно безліч ботів з найрізноманітнішим функціоналом. Звичайно в такому підході є і свої недоліки:

- При зростанні кількості чатів з ботами стає складно знаходити їх, нагромаджується загальний список чатів.
- Кожен бот створюється окремим розробником, можливі витоки персональних даних (Наприклад ситуація з ботом “Get contact”)

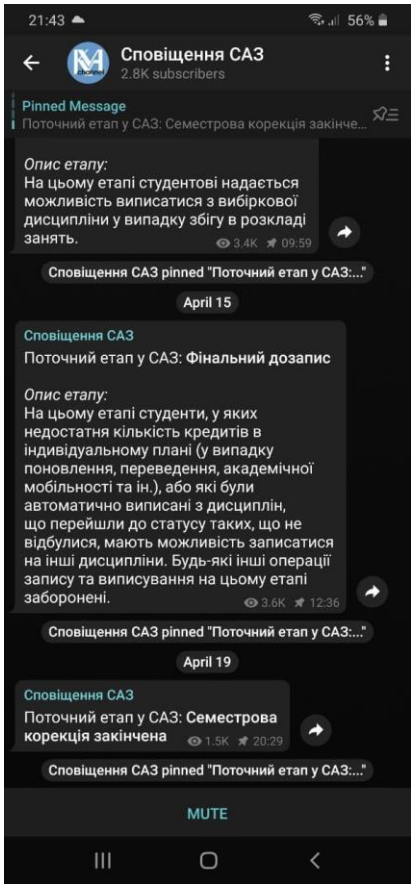


Рисунок 1.1 – Телеграм бот для сповіщення початку запису на дисципліни в НАУКМА

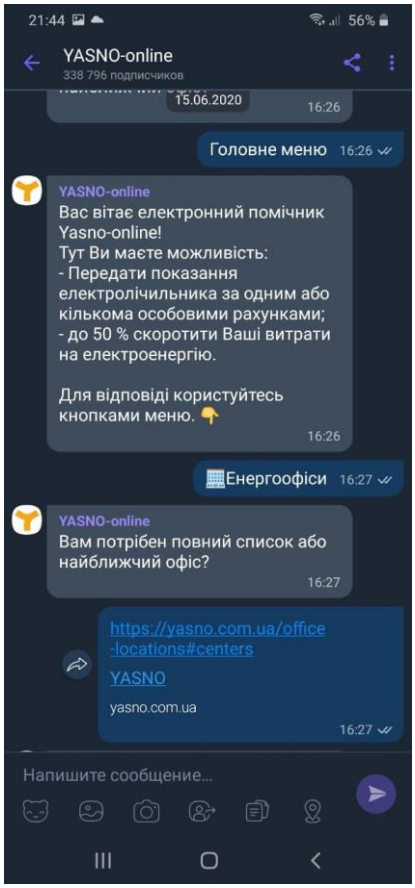


Рисунок 1.2 – Бот у Viber для оплати комунальних послуг

Facebook M – тестовий проект компанії Facebook ціль якого спростити життя користувачам з повсякденними задачами. Перевагою цього боту була можливість розпізнавати вільний текст від користувача. Застосунок міг: резервувати столики в ресторані, планувати поїздки, робити покупки онлайн. Для цього продукту можна виділити такі недоліки:

- Це був тестовий проект з обмеженим доступом від Facebook і він був закритий через 2.5 роки(2015 - 2018)
- Підозри щодо використання отриманих фейсбуком персональних даних задля реклами/маркетингу
- Бот виступав в ролі окремого застосунку що ускладнювало його використання разом з месенджером.

Viber – також дуже популярний месенджер особливо на території України. В останні роки стрімко зростає та не так давно в ньому був введений функціонал ботів аналогічний до ботів Telegram. Виявлені недоліки аналогічні недолікам Telegram ботів.

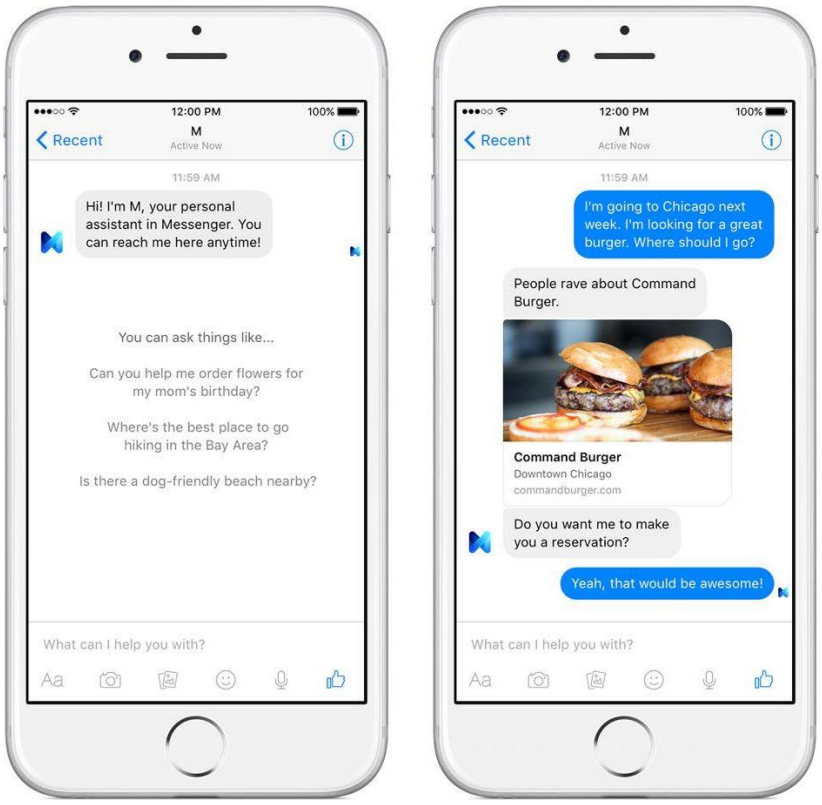


Рисунок 1.3 – Facebook M асистент [1]

1.3 Постановка завдання

Отже результатом розробки буде застосунок для мобільних пристроїв. Цей застосунок надаватиме функціонал месенджеру з вбудованим помічником для виконання певних команд. Також потрібно розробити сервер та підключити його до бази даних в якій буде зберігатись інформація про користувачів, чати, дані які запам’ятовуватиме асистент про кожного користувача.

Можна виділити такий базовий функціонал:

- Реєстрація нових користувачів
- Аутентифікація користувачів

- Можливість автоматичного входу при бажанні користувача
- Можливість пошуку інших користувачів
- Створення приватних чатів
- Створення групових чатів з 2 ролями (адміністратор та звичайний користувач)
- Дії в публічних чатах: передача прав адміністратора, додавання нових учасників, видалення учасників, можливість покинути чат.
- Контент повідомлень: текст, зображення(зовнішні), посилання, OG теги.
- Видалення певних повідомлень для користувача назавжди
- Вихід з аккаунту

Функціонал віртуального асистенту:

- Ведення простого діалогу з користувачем
- Надання інформації про погоду в заданому місті
- Встановлення нагадувань на певний час
- Надання новин у вигляді OG тегів
- Можливість вибору або зміни джерела новин
- Ведення списку завдань користувача (ToDo list)

Вимоги до застосунку:

- Безпека передачі даних по мережі
- Безпечне збереження конфіденційної інформації (паролів) у базі даних
- Локальне збереження даних для зменшення навантаження на сервер при оновленнях.
- Інтуїтивний інтерфейс.
- Прийнятна швидкість доставки повідомлень в реальному часі
- Можливість розширення серверу для роботи з веб та десктоп клієнтами.

1.4 Висновки до розділу 1

В даному розділі була розглянута актуальність розробки месенджеру з віртуальним асистентом також висвітлено проблеми які такий додаток покликаний вирішити. На основі приведених прикладів було зроблено висновок який формат асистенту є кращим. В кінці розділу було чітко визначено майбутній функціонал як месенджеру так і самого асистенту.

Розділ 2. Дослідження та теоретичний аналіз серверних технологій

2.1 Опис серверних технологій

2.1.1 Мова програмування C# та платформа .NET Core

C# - об’єктно-орієнтовна мова програмування розроблена в 1998- 2001 роках компанією Microsoft. C# - статично типізована мова, має C-подібний синтаксис, схожа на мови Java та C++. Виконання програм відбувається у середовищі CLR (Common Language Runtime) яке є спільним для усіх .NET сумісних мов програмування.

.NET Core – модульна платформа для розробки програмного забезпечення. До неї входять стандартна бібліотека CoreFX для роботи з колекціями, потоками вводу та виводу. Також в платформу входить середовище виконання CLR. .NET Core надає зручний менеджер пакетів – NuGet, який дозволяє встановлювати додаткові бібліотеки, залежності.

2.1.2 NoSQL база даних - MongoDB

MongoDB – документно-орієнтовна система керування базами даних. Не потребує опису схем таблиць, зберігає дані у JSON-подібному форматі. До переваг можна віднести: відносну швидкість, гнучкість мови запитів, легкість модифікації існуючих документів.

Важливо згадати сумісність MongoDB та мови C#. Розробниками поширюється драйвер з відкритим вихідним кодом для доступу до MongoDB з коду на C#. Значною перевагою цього драйвера є можливість створення запитів за допомогою технології LINQ, що спрощує тестування, запобігає можливим синтаксичним помилкам, дозволяє розробнику швидше пристосуватись до нової технології.

2.2 Вибір серверної архітектури

В результаті аналізу можливих варіантів було обрано багаторівневу проектну архітектуру[2]. Це зумовлено великою кількістю операцій які потрібно

застосовувати на даних перед їхнім збереженням в базі даних або відправці клієнту. Кожний рівень представлений окремим проектом який надає вузький інтерфейс використання та повністю приховує у собі логіку певного рівня.

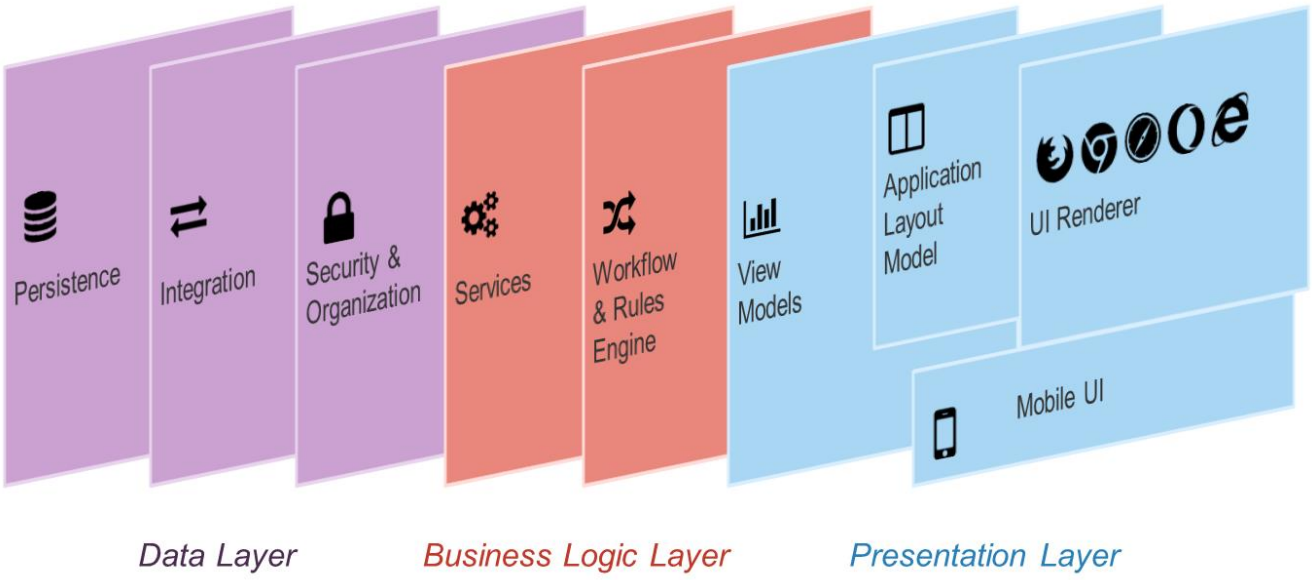


Рисунок 2.1 – Приклад багаторівневої архітектури

Проектний стиль розробки обраний з огляду на майбутній ріст та розширення серверу. Завдяки окремим проектам-модулям можна компілювати та оновлювати файли серверу частинами тому що результатом компіляції кожного проекту є один файл з розширенням «.dll» та декілька конфігураційних файлів. Також це значно пришвидшує розробку якщо в командні працюють багато програмістів.

2.3 Проектування мережевої взаємодії

2.3.1 Вибір базового протоколу

Передача інформації по мережі відбувається за допомогою мережевих протоколів. Через велику кількість різних задач при передачі даних сформувався так званий рівневий підхід до роботи з даними у мережі. У 1978 році була

сформована абстрактна мережева модель протоколів – модель OSI яка зображена на рисунку 2.2. В ній описується кожний рівень мережної взаємодії [3].

Кожному рівню відповідають деякі існуючі протоколи які виконують задачі на цьому рівні. Перші три рівні – прикладний, представлення та сеансу зазвичай працюють з самими даними тільки на рівні застосунку. Транспортний рівень – вже відповідає за те як буде відбуватись обмін даними між різними пристроями.

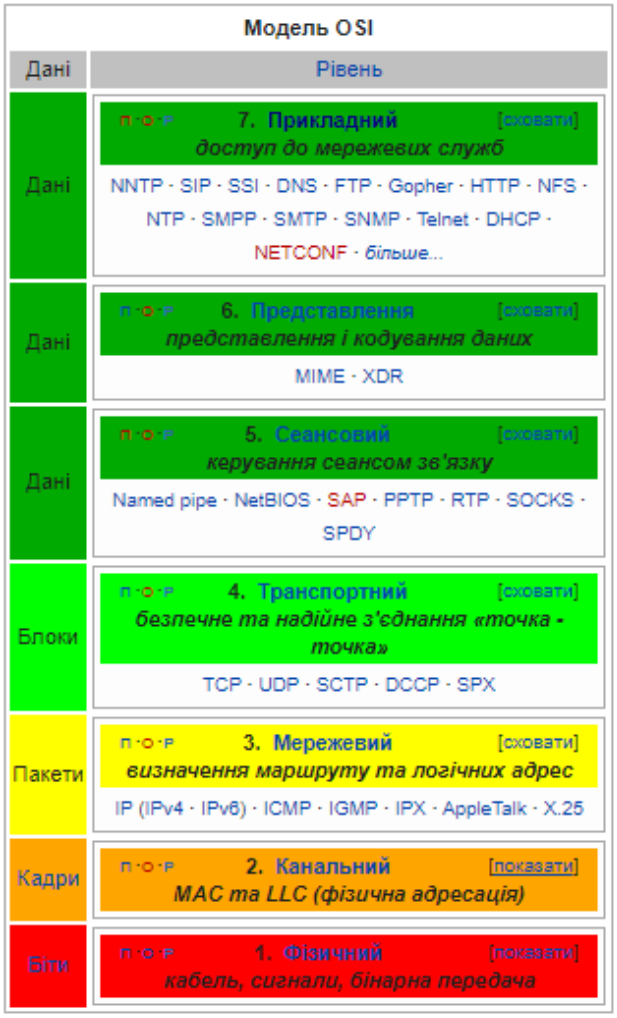


Рисунок 2.2 – модель OSI

Отже почати треба саме з вибору протоколу транспортного рівня. Зазвичай вибір роблять між двома основними транспортними протоколами TCP або UDP.

TCP – протокол керування передачею. Його основна задача забезпечити надійний обмін даними між пристроями. Цей протокол гарантує що дані будуть доставлені та вони не будуть пошкоджені після доставки. Для досягнення таких результатів TCP веде нумерацію кожного сегменту даних який передається по мережі і також рахується та передається контрольна сума для перевірки цілісності. Звичайно такий підхід потребує передачі додаткових даних і TCP сегменти через це можуть бути доволі великими.

UDP – протокол датаграм користувача. Його основна властивість в тому що він не гарантує доставку та цілісність даних. Він набагато простіший від TCP, що в свою чергу робить його швидшим і не потребує передачі додаткових даних.

UDP можуть використовувати в відеотрансляціях, голосових чатах, тобто при передачі потокових даних і в разі невеликих спотворень нам не потрібно досилати втрачені дані. TCP – використовується при передачі повідомлень - даних які не повинні бути пошкоджені або втрачені. Можна зробити висновок що для реалізації месенджеру необхідний надійний протокол який точно доставить наші дані – тобто TCP.

2.3.2 Взаємодія з великою кількістю клієнтів

Важливою задачею сервера є одночасне отримання, обробка та відправка даних багатьом користувачам. Такої паралельності можна досягти використовуючи потоки виконання (threads). Кожному потоку виділяється квант процесорного часу, після чого процесор швидко перемикається між кожним потоком поступово виконуючи задачі.

Виділення потоку для задачі може бути доволі обтяжливою дією для операційної системи. Особливо багато часу може зайняти ініціалізація нового потоку. Щоб збільшити швидкодію потрібно використовувати шаблон «Пул потоків» (Pool of threads)[4]. Основна ідея такого підходу в тому, що потоки ініціалізуються тільки один раз і надалі існують весь час роботи програми. Якщо потрібно виконати якусь роботу паралельно – задача передається одному з потоків пулу. Після виконання задачі потік знову повертається в пул.

Платформа .NET Core надає зручний клас для роботи з вже реалізованим пулом потоків – System.Threading.ThreadPool. Кількість потоків в такому пулі обмежена тільки адресним простором виконуючої машини. Для мого пристрою такий ліміт – 32767 потоків.

Коли клієнт встановлює з’єднання з сервером - з пулу потоків виділяється один для роботи з цим клієнтом. Надалі цей потік не повернеться до загального пулу доки клієнт не від’єднається. Враховуючи попередні перевірки можна зробити висновок що мій пристрій може працювати одночасно з 32767 клієнтами.

2.3.3 Відправка та отримання даних

Так як було вирішено використовувати протокол TCP, потрібно обрати відповідні класи для взаємодії з клієнтом. Для створення об’єкту TCP з’єднання був обраний клас TcpClient з вбудованої бібліотеки System.Net.Sockets. Перевагою класу є можливість відправляти пакети для перевірки статусу з’єднання клієнту. В разі розриву підключення - обробити відповідне виключення та коректно завершити роботу потоку.

Для взаємодії з даними був використаний клас NetworkStream. Він дозволяє отримувати та відправляти дані клієнту у вигляді потоку байтів.

2.4 Проектування власного протоколу

2.4.1 Проектування вмісту протоколу

В описі мережевого модуля був обраний протокол транспортного рівня – TCP. Але це лише один з рівнів моделі OSI (рисунок 2.2), як на рахунок інших рівнів? Мережевий, канальний та фізичний рівні це турбота самих пристроїв та вбудованого ПЗ. Залишаються рівні - прикладний, представлення та сеансу. Часто ці рівні можна об’єднати в один загальний рівень – рівень застосунку. На ньому відбувається шифрування, кодування, представлення даних. Для опрацювання цих задач було вирішено реалізувати власний протокол.

По-перше, головна мета протоколу – передача даних, отже потрібно вирішити в якому форматі їх передавати. Найпопулярнішими форматами на сьогодні є JSON та XML [5].

JSON (запис об’єктів JavaScript) – формат представлення даних який з’явився на початку 2000-х років. Через простий синтаксис дає змогу зручно описувати об’єкти та структури даних. Дані зберігаються в форматі «ключ-значення». Значною перевагою є сумісність з JavaScript та простота його синтаксичного розбору в цій мові.

XML (розширювана мова розмітки) – з’явився в 1998 році. Дані зберігаються в середині тегів, теги також можуть мати атрибути з допоміжними даними. Більш об’ємний по розміру ніж JSON. Також підтримується в мові JavaScript.

Отже, з огляду на простоту використання та розмір, формат JSON був обраний для представлення даних в нашому протоколі.

Зручною особливістю була б можливість дізнатись призначення відправленого пакету ще до перетворення даних в потрібний формат. Для цього до пакету даних можна приєднати додатковий головний байт який і буде нести в собі потрібну інформацію про призначення пакету.

Однією з особливостей TCP є розбиття даних на сегменти, причому ці сегменти не будуть відповідати пакетам вищого рівня - в даному випадку пакетам нашого протоколу. Тож необхідно якось виокремлювати пакети рівня застосунку з потоку байтів який буде отриманий по транспортному протоколу. Для цього можна додавати особливий байт у кінці пакету, який буде однаковий в усіх пакетах. По ньому можна зрозуміти що клієнт передав один цілий об’єкт інформації, і тепер можна передавати ці дані на інші рівні серверу.

Графічно можно представити вид пакету протоколу так як показано на рисунку 2.3.

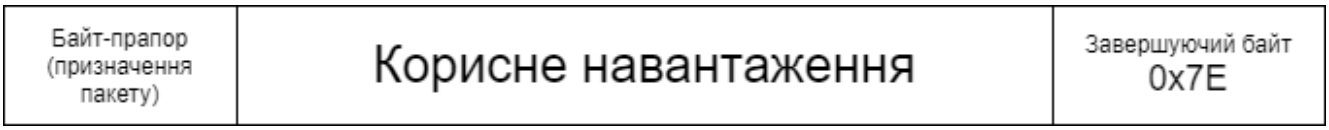


Рисунок 2.3 - схема пакету

2.4.3 Безпека передачі даних

Важливо подбати про безпеку даних які передаються мережею. По-перше треба вирішити який алгоритм шифрування буде використовуватись. Існує два типи алгоритмів шифруванні – симетричні та асиметричні [6] .

Симетричні алгоритми шифрування використовують один і той же ключ для шифрування та дешифрування даних. Воно швидко працюють, але перед початком

роботи цього алгоритму потрібно якось встановити загальний ключ між двома співрозмовниками. Розповсюджені приклади таких алгоритмів: DES, AES, SEAL, RC.

Асиметричні алгоритми шифрування використовують пару ключів згенерованих спеціальним чином. Один з цих ключів – приватний, тобто не передається по мережі та за допомогою нього розшифровуються дані, інший – публічний, його можна передавати по мережі і ним шифруються дані. Особливістю є також те що маючи тільки публічний ключ - відповідний приватний вирахувати за прийнятний час неможливо. Для передачі повідомлення один з користувачів запитує в іншого публічний ключ, отримує його і зашифровує ним своє повідомлення і далі передає по мережі. Другий користувач застосовує свій приватний ключ для розшифровки даних. Ці алгоритми набагато повільніші за симетричні алгоритми шифрування. Приклади таких алгоритмів: Diffie-Hellman, RSA, ECC.

Через свою складність та повільність асиметричні алгоритми використовуються для обміну ключами а для самого обміну повідомленнями використовують симетричні алгоритми завдяки їхній швидкості. Так і було вирішено організувати шифрування в нашому протоколі. А саме – після встановлення TCP сесії клієнт з сервером запускають алгоритм Diffie-Hellman і в результаті вони домовляються про спільний ключ для шифрування та дешифрування даних. Після цього будь-які дані шифруються за допомогою симетричного алгоритму AES-256 та встановленого спільного ключа. Реалізації алгоритмів AES-256 та Diffie-Hellman були використані з вбудованої бібліотеки System.Security.Cryptography.

2.5 Вибір бази даних та основи роботи з нею

2.5.1 Реляційна та нереляційна СКБД

На сьогоднішній день при розробці клієнт-серверних застосунків при виборі СКБД розглядають два варіанти – реляційні та нереляційні підходи [7].

Реляційні СКБД для зберігання даних використовують таблиці з рядками та стовпцями. Основна вимога такого підходу – заздалегідь визначена схема таких

таблиць. Тобто – які стовпці існують у певній таблиці та які типи даних можуть у них зберігатись. Для зв’язку таблиць використовують унікальні ключі в одній таблиці, та зовнішні ключі в інших таблицях, значення яких співпадають при наявності зв’язку між певними рядками. Для роботи з даними - їх отриманню, видаленню та додаванню використовують мову запитів – SQL.

Нереляційних підходів існує багато, але один з найпопулярніших це – документо-орієнтовна модель. Основна відмінність від реляційної в тому, що схеми документів можна не визначати заздалегідь і поля документів однієї колекції можуть відрізнятись кількістю, іменами, типами значень. Формат зберігання даних в таких БД – JSON подібний . Зв’язок між документами можна організувати таким же чином як і в реляційній моделі або використати вкладені документи, що також може спростити роботу з даними.

Враховуючи гнучкість другого підходу та простоту модифікації схем зберігання даних було вирішено використати нереляційну модель, а саме – документо-орієнтований підхід.

2.5.2 СКБД MongoDB

З існуючих документо-орієнтованих СКБД було обрано MongoDB[8]. З переваг даного продукту було визначено:

- Наявність гнучкої, функціональної мови запитів
- Висока продуктивність
- Наявність MongoDB драйверу для зручної роботи з БД на мові C#
- Програмне забезпечення MongoDB Compass для перегляду, модифікації, видалення даних у графічній оболонці
- Передбачений доступ до БД з багатопоточних додатків
- Підтримка в багатьох операційних системах (особливо Windows та Debian) та швидкість її налаштування в них.

2.5.3 Шаблони для роботи зі сховищем даних

В складних проектах при роботі з базами даних необхідно структурно організувати код взаємодії з даними, потрібно заздалегідь передбачити зручний інтерфейс доступу до такого коду та зробити його зручним для модифікації. Такий код називають «Прошарок доступу до даних» або DAL. Такі вимоги вплинули на появу абстрактних шаблонів, наприклад шаблони DAO та Репозиторій[9] які дозволяють підійти до розробки більш модульно.

DAO – об’єкт доступу до даних. Його особливість в тому що кожній сутності відповідає тільки один DAO-об’єкт. Мета DAO в першу чергу імплементувати CRUD операції (збереження, зчитування, оновлення, видалення). При необхідності в певному DAO можна реалізувати специфічні методи, які потрібні при роботі тільки з конкретною сутністю. Наприклад: якщо в проекті існує клас-сутність User то для його взаємодії з базою даних необхідно реалізувати клас UserDAO.

Репозиторій – підхід при якому користувач взаємодіє з сутностями певного типу як з цілою колекцією. Типовими методами класу репозиторій є додавання, видалення, перевірка наявності за певними параметрами. Враховуючи що DAO працює з сутностями поодинці можна побудувати репозиторій використовуючи DAO. Часто ці шаблони комбінують і те що на початку проектувалось як репозиторій в кінцевому результаті можу бути ближчим до DAO.

Враховуючи що при розробці месенджеру часто доведеться працювати з цілими масивами даних, наприклад: масив чатів, масив повідомлень чату, списки користувачів - було вирішено притримуватись шаблону репозиторій для реалізації прошарку доступу до бази даних. Незважаючи на це, все одно довелось реалізувати деякі методи які більш властиві DAO, наприклад оновлення полів певного документу.

2.6 Висновки до розділу 2

В даному розділі було обрано та досліджено технології для розробки серверної частини додатку. Обраний спосіб передачі даних та забезпечення їх

конфіденційності під час транспортування по мережі. Були розглянуті підходи до роботи зі сховищем даних та обрана технологія бази даних.

Розділ 3. Дослідження та теоретичний аналіз фронтенд технологій

3.1 Загальний опис фронтенд технологій

3.1.1 Фреймворк React Native та мова Javascript

React Native – фреймворк для розробки мобільних або настільних додатків, випущений компанією Facebook у 2015 році. Підтримує багато платформ серед яких: Android, IOS, Web, Windows. Принцип роботи React Native аналогічний принципу роботи веб-фреймворку React, але при цьому не використовується об'єктна модель документу (DOM), замість неї React Native використовує інтерфейсні компоненти платформи на якій виконується. Фреймворк дозволяє писати функціональні компоненти для представлення частин інтерфейсу програми. Основна мова програмування – Javascript. Також фреймворк дозволяє писати модулі на інших мовах, які підтримуються платформою. Наприклад для операційної системи Android – на Java або Kotlin, для IOS – Swift або Objective-C.

Javascript – нетипізована мова програмування, випущена в 1995 році. Початковим призначенням мови було її використання в браузерах для вбудовування в HTML сторінки. Згодом, з розвитком мови стало можливо писати на Javascript серверні застосунки. В фреймворку React Native мова Javascript є лише зручним інструментом розробника, тому що цей код не виконується інтерпретатором як у веб-браузерах, він компілюється в нативні модулі на тій мові яка є стандартною для платформи.

3.1.2 Бібліотека Redux

Ця бібліотека використовується з фреймворками React, React Native, Angular. Її основне призначення – створення сховища станів додатку та керування зміною цих станів. Стан в React це набір даних – змінних та функцій які можуть зберігатись між оновленнями компонентів. Причина створення такої бібліотеки – складність

керування станами додатку через функціональний характер фреймворків на зразок React.

Послідовність виконання зміни стану за допомогою цієї бібліотеки можна описати так:

1. Користувач виконує якусь дію в графічному інтерфейсі
2. Виконується код який запускає дію з певними параметрами. Дія представляє собою об’єкт з певними ключами та значеннями.
3. Створений об’єкт отримує Reducer, це функція яка в залежності від значення поля type виконає якусь дію та зробить зміну стану.
4. Повернений стан встановлюється як новий стан додатку.

Ілюстрація цих дій зображена на рисунку 3.1.

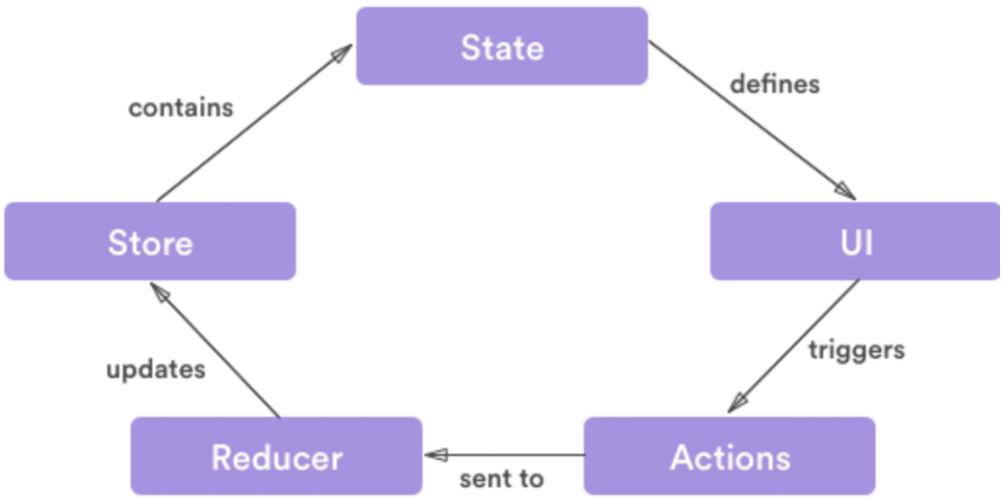


Рисунок 3.1 - Цикл подій у redux

3.2 Обґрунтування використання Java модулів

3.2.1 Опрацювання даних отриманих з мережі

Основна мова програмування для операційної системи Android це Java. Враховуючи те, що додаток має працювати на цій OS додаткові нативні модулі були написані саме на цій мові.

Перша проблема виникає при роботі з даними мережі – вони отримуються як суцільний масив байтів які треба правильно обробити. Для такої задачі більше підходить строго типізована мова програмування - це дозволяє уникнути можливих помилок та спрощує тестування в процесі розробки.

3.2.2 Алгоритм обміну ключами

З опису серверної частини можна зробити висновок що для встановлення загального ключа шифрування між сервером та клієнтом використовується асиметричний алгоритм шифрування – алгоритм діффі-хелмана. За результатами пошуку та аналізу бібліотек для фреймворку React Native було виявлено що ні одна бібліотека не реалізує потрібний алгоритм. Це пов’язано з тим що зазвичай розробники використовують готові, перевірені часом протоколи з підтримкою шифрування, наприклад HTTPS. В нашому випадку протокол самописний і для реалізації алгоритму можна написати модуль на Java використовуючи бібліотеки цієї мови. Такі бібліотеки як javax.crypto та java.security підійдуть для даної задачі.

3.2.3 Робота з шифруванням

Після успішного встановлення загального ключа шифрування між сервером та клієнтом починається обмін даними, причому дані шифруються симетричним алгоритмом AES-256. Для імплементації цього алгоритму також можна використати бібліотеки javax.crypto та java.security.

3.3 Дослідження компонентної системи React Native

3.3.1 Загальні відомості

Обраний фреймворк забезпечує програміста базовими компонентами[10] які можна використовувати для розробки додатку. Наразі доступні компоненти для роботи з кнопками, текстом, картинками, модальними вікнами, індикаторами завантаження та прогресу, компонентами для відображення прокручуваних списків.

Кожний компонент має свої атрибути які можуть бути вказані розробником, або в протилежному випадку автоматично взяті за замовчуванням. Значенням

атрибутів можуть бути функції, стрічки, числа, масиви і навіть цілі об’єкти. Одним з атрибутів значення якого можна вказати в усіх компонентах це атрибут style. Даний атрибут дозволяє вказати графічний вигляд компоненту за допомогою мови стилів – CSS.

3.3.2 ScrollView проти FlatList

При розробці месенджеру частою задачею є виведення списку елементів які користувач може прокручувати. Наприклад: потік повідомлень у чатах, списки чатів, списки користувачів і також сторінки які не вміщуються по вертикалі екрану. Для цього React Native надає декілька готових рішень.

ScrollView – простий прокручуваний список. При роботі з ним достатньо загорнути в нього потрібні елементи. Цей компонент надає мінімальне управління вмістом, та завантажує для відображення одразу всі елементи. Підходить для відображення невеликих сторінок.

FlatList – прогресивна версія ScrollView. Головна особливість – підтримка лінивого завантаження, тобто вміст завантажується по мірі необхідності. Такий підхід дозволяє збільшити швидкодію, оптимізувати роботу додатку. Також цей компонент дозволяє робити розділення списку, підтримує декілька стовпців, дозволяє “потягнути” список для дії оновлення, прокручувати список автоматично до певного елементу. Призначення компоненту – відображення дуже довгих списків елементів.

Можна зробити висновок, що для прокрутки простих сторінок з чатами, відображення сторінки з налаштуваннями або реєстрацією можна використати компонент ScrollView. Але при виведенні повідомлень чатів які зазвичай можуть досягати десятків тисяч елементів краще застосувати FlatList.

3.3.3 Бібліотека розширення “React Native Elements”

Компоненти з фреймворку React Native надають дуже загальний функціонал. При створенні месенджеру можуть знадобитись більш спеціалізовані компоненти. З такою задачею гарно справляється бібліотека для розширення стандартних

компонентів під назвою React Native Elements[11]. З корисних компонентів що стосуються розробки месенджерів та соціальних мереж можна виділити елементи для відображення аватарів користувачів, чатів, плаваючі кнопки, піктограми, підказки, картки для даних. Це значно пришвидшує розробку візуальної частини та зменшує кількість коду.

3.4 Використання бібліотеки Redux

3.4.1 Модуль для роботи з мережею

Щоб зробити доступ до дій з мережею зручним було вирішено помістити їх в окремий редьюсер. В ньому відбуваються дії відправки та отримання даних. Також в ньому використовуються нативні модулі на Java для шифрування та дешифрування даних, опрацювання та формування мережевих пакетів.

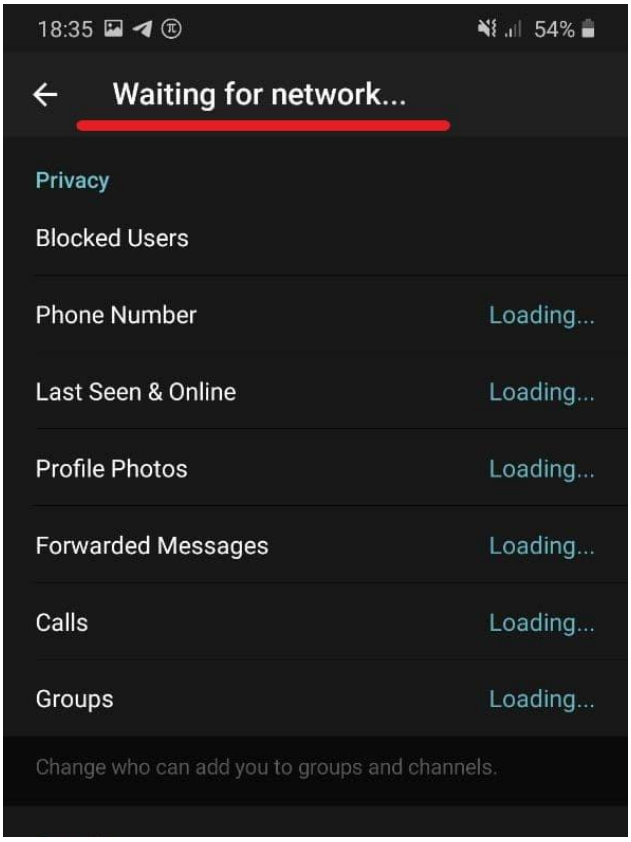
При відправці даних з серверу необхідно відправляти ці дані певним компонентам. Для цього реалізована система підписок, що дозволяє будь-якому компоненту підписатись на оновлення з сервера і отримувати дані як тільки вони надходять клієнту.

3.4.2 Модуль для роботи з локальним сховищем

Аналогічним чином організована робота з локальним сховищем даних. Є окремий редьюсер що виконує такі дії як ініціалізація об’єкту доступу до сховища, реалізовує операції додавання, оновлення, зчитування та видалення даних зі сховища.

3.4.3 Модуль для відображення мережевих помилок

Ще однією перевагою бібліотеки є можливість зміни візуального стану додатку. Якщо розглянути типові месенджери такі як Telegram, Viber, Facebook Messenger то кожний з них має свою систему сповіщення користувача про помилки. Особливо важливими є повідомлення про розрив з’єднання з сервером.



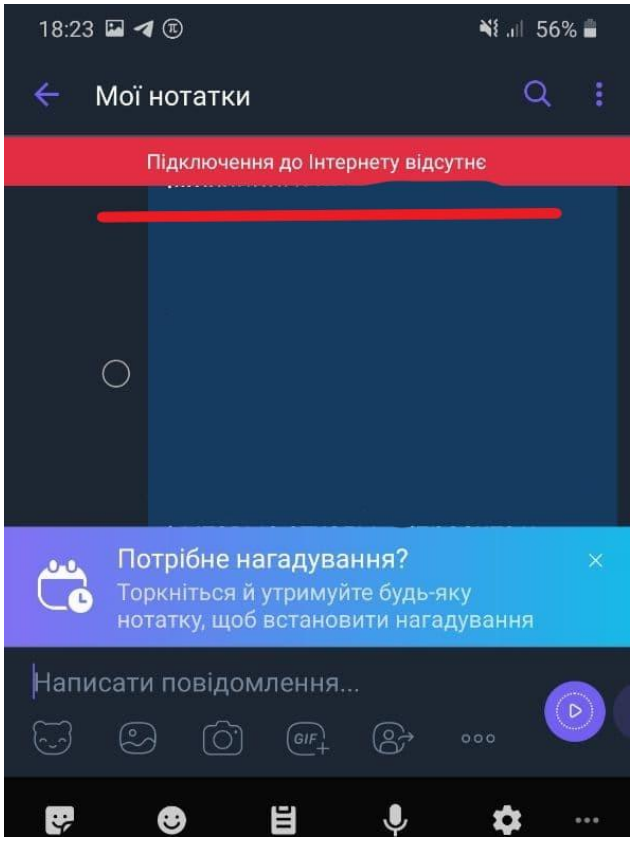


Рисунок 3.2 - розрив з'єднання Telegram Рисунок 3.3 - розрив з'єднання Viber

На рисунках 3.2 та 3.3 можемо побачити приклади таких повідомлень. Ці повідомлення можуть бути відображені в будь-якому стані додатку, наприклад коли користувач знаходиться в чаті, на сторінці списків чатів, при перегляді налаштувань, тощо. Отже в даному випадку можна використати саме redux для відображення таких компонентів для того щоб користувач був сповіщений вчасно і в будь-якому стані додатку.

3.5 Висновки до розділу 3

В даному розділі було обрано та досліджено технології для розробки клієнтського додатку. Обґрунтоване використання додаткових Java модулів та технології Redux. Також були розглянуті додаткові бібліотеки для швидкої розробки візуальної частини додатку.

Розділ 4. Опис реалізації програмного продукту

4.1 Реалізація серверу для додатку

4.1.1 Загальний опис серверних модулів

Так як була обрана багаторівнева архітектура то кожен рівень можуть представляти декілька серверних модулів. Призначення модулів можна дізнатись з рисунку 4.1.

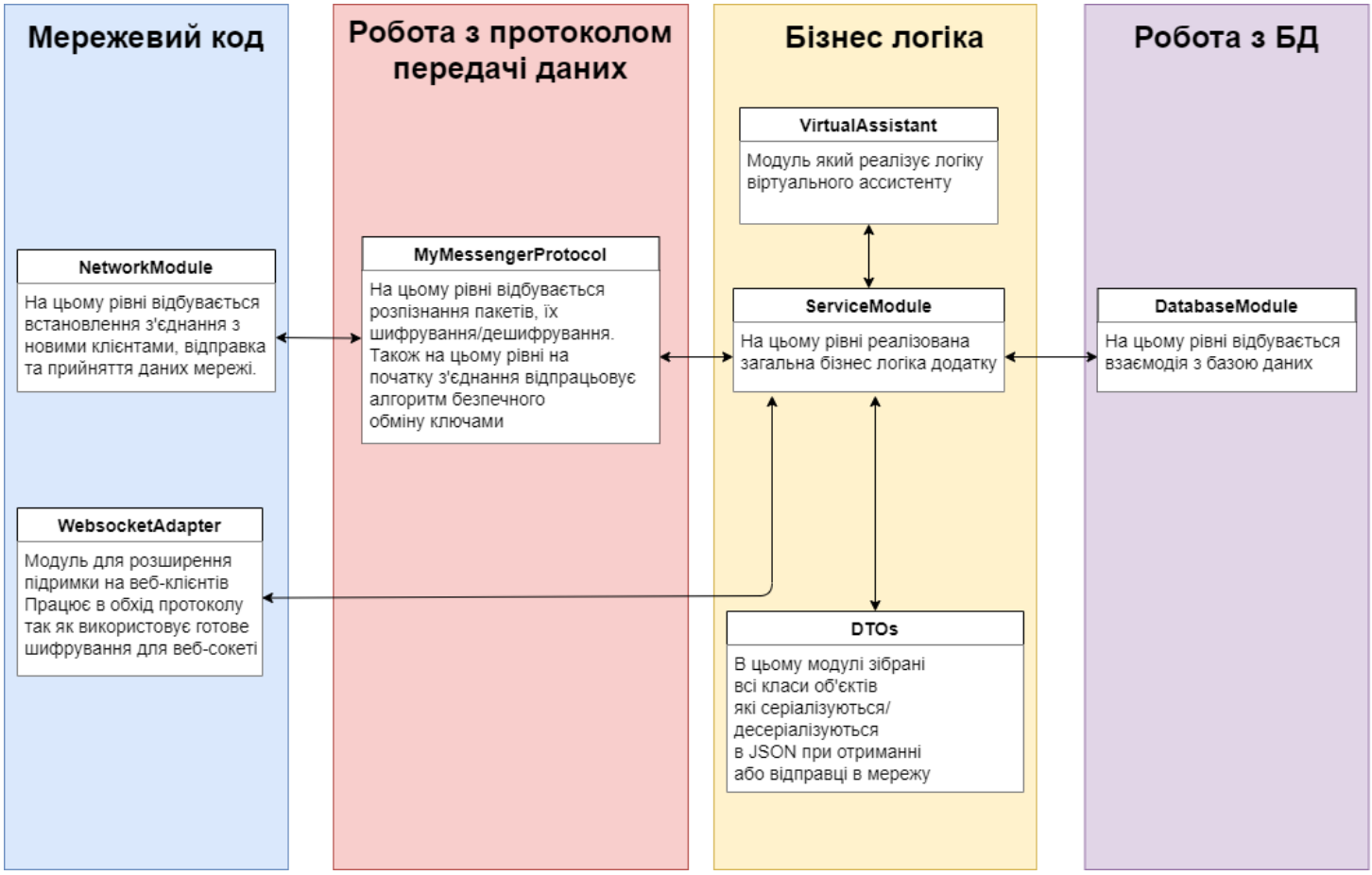


Рисунок 4.1 - Опис модулів та їх приналежність до рівнів

4.1.2 Реалізація NetworkModule

Цей модуль складається з таких класів зображених на рисунку 4.2

Рисунок 4.2 - Діаграми класів NetworkModule.cs

Program.cs – точка входу в програму, тут створюється об’єкт класу MainListener.cs і передаються параметри айпі адреса та порт на якому сервер буде працювати.

MainListener.cs – клас який відповідає за прослуховування вказаного порту на наявність нових клієнтів. В його методі Listen відбувається нескінченний цикл, при кожній ітерації якого виконання зупиняється допоки не з’явиться новий клієнт. При підключенні клієнта відбувається створення об’єкту класу ClientObject.cs і одному з потоків пулу передається задача на виконання методу Process.

ClientObject.cs – відповідає за роботу з одним клієнтом. Головний метод тут це Process в якому відбувається прийняття або відправка даних клієнту. Також у

класі є делегати SendUpdate та UserLoggedIn. Ці делегати передаються нижчим рівням серверу для роботи з відправкою оновлень чатів в реальному часі.

4.1.3 Реалізація MyMessengerProtocol

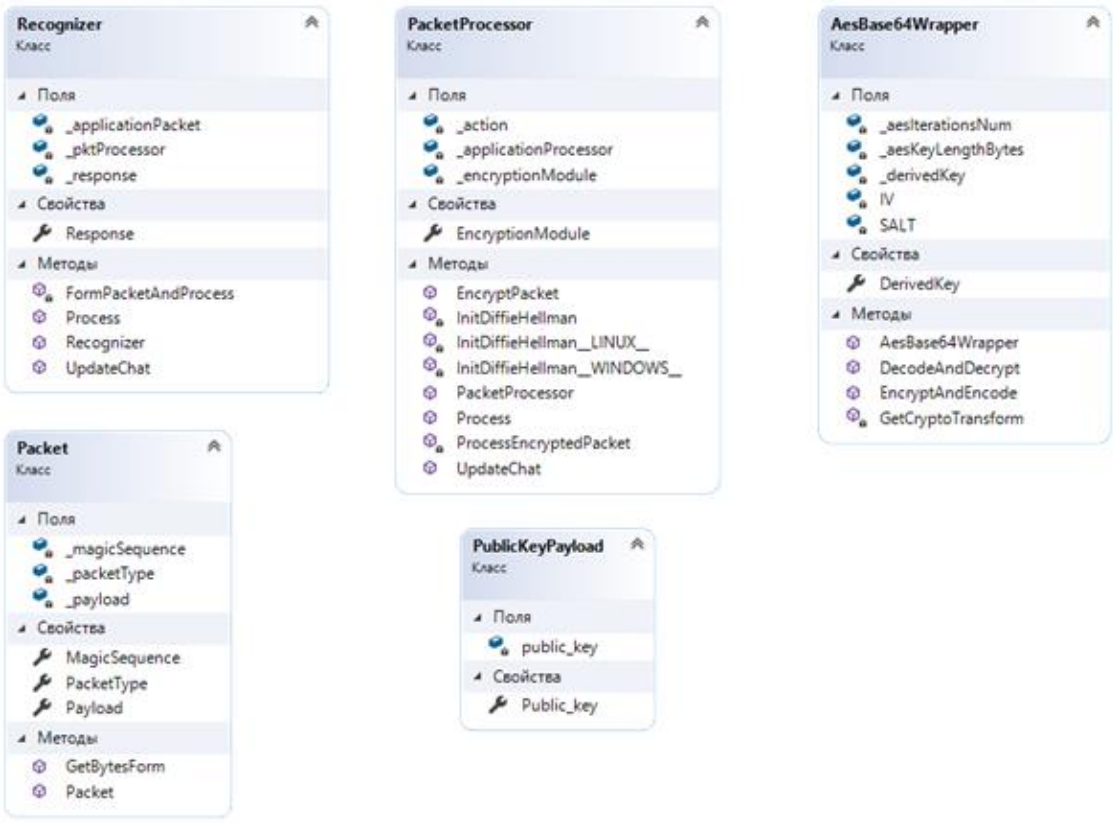


Рисунок 4.3 - Діаграми класів MyMessengerProtocol.cs

Recognizer.cs – клас в якому відбувається розпізнання пакетів протоколу, поміщення даних в структуру Packet.cs і його передача до обробника даних пакету – класу PacketProcessor.cs.

PacketProcessor.cs – клас в якому розміщені основні механізми безпеки даних пакету. Також тут міститься алгоритм встановлення ключа шифрування – алгоритм діффі-хелмана. Він реалізований у двох екземплярах – для windows та для linux, тому що в різних операційних системах алгоритм надається різними бібліотеками. При звичному обміні повідомленнями використовується клас AesBase64Wrapper.cs який і використовує встановлений до цього ключ для шифрування та дешифрування даних.

AesBase64Wrapper.cs – використовується для застосування шифрувального алгоритму AES-256 та кодування даних за алгоритмом Base64 для зручного представлення бінарних даних в текстовому форматі.

4.1.4 Реалізація ServiceModule

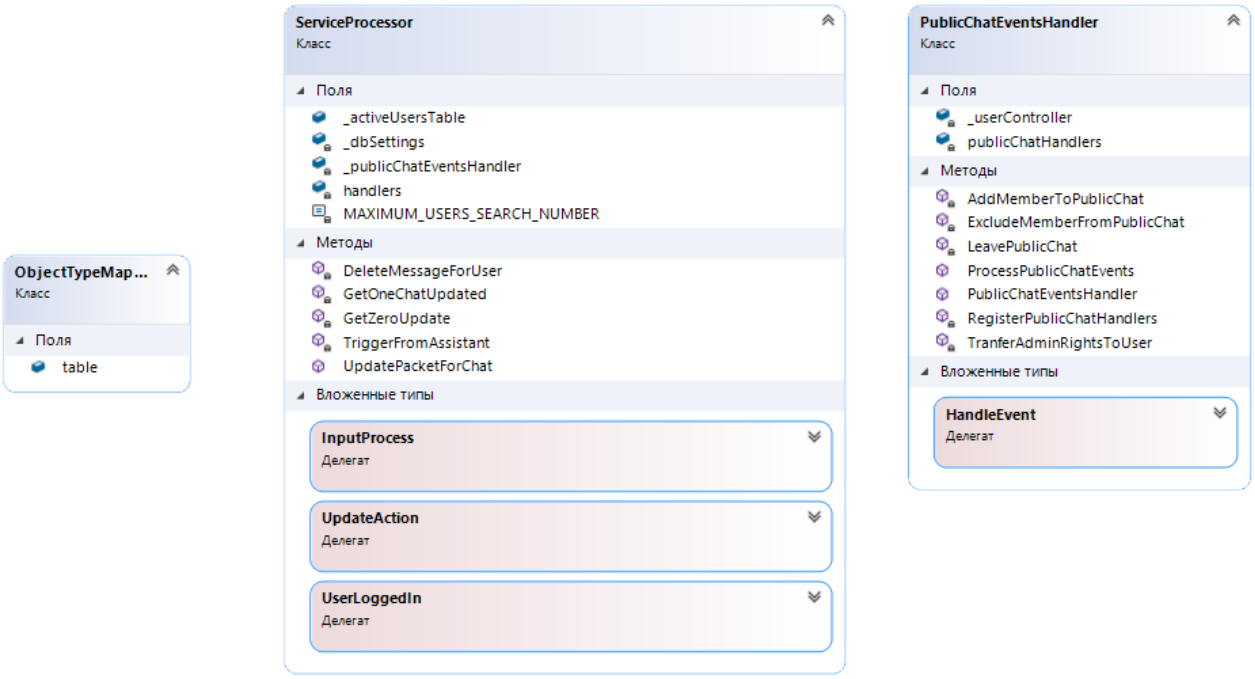


Рисунок 4.4 – Діаграми класів ServiceModule.cs

SericeModule.cs – містить методи які виконуються при різних діях користувача. ObjectTypeMapper.cs – клас який прив’язує типи пакетів до відповідних методів обробки. Також тут відбувається використання делегатів переданих від NetworkModule.cs. Важливим елементом є поле _activeUsersTable – це потокобезпечна таблиця ключів значень, де ключами є ідентифікатори користувачів а значеннями делегати для їхнього повідомлення про оновлення певних чатів. Коли користувач заходить у свій аккаунт то він вноситься в цю таблицю. Також модуль веде звіт про останні дані які були надіслані користувачу і в разі оновлення відправляє клієнту тільки ті дані яких в нього не було.

PublicChatEventsHandler.cs – допоміжний клас в якому відбувається обробка усіх подій групових чатів – додавання користувачів, виключення користувачів, покидання чату, передача адмінський прав.

4.1.5 Реалізація DatabaseModule

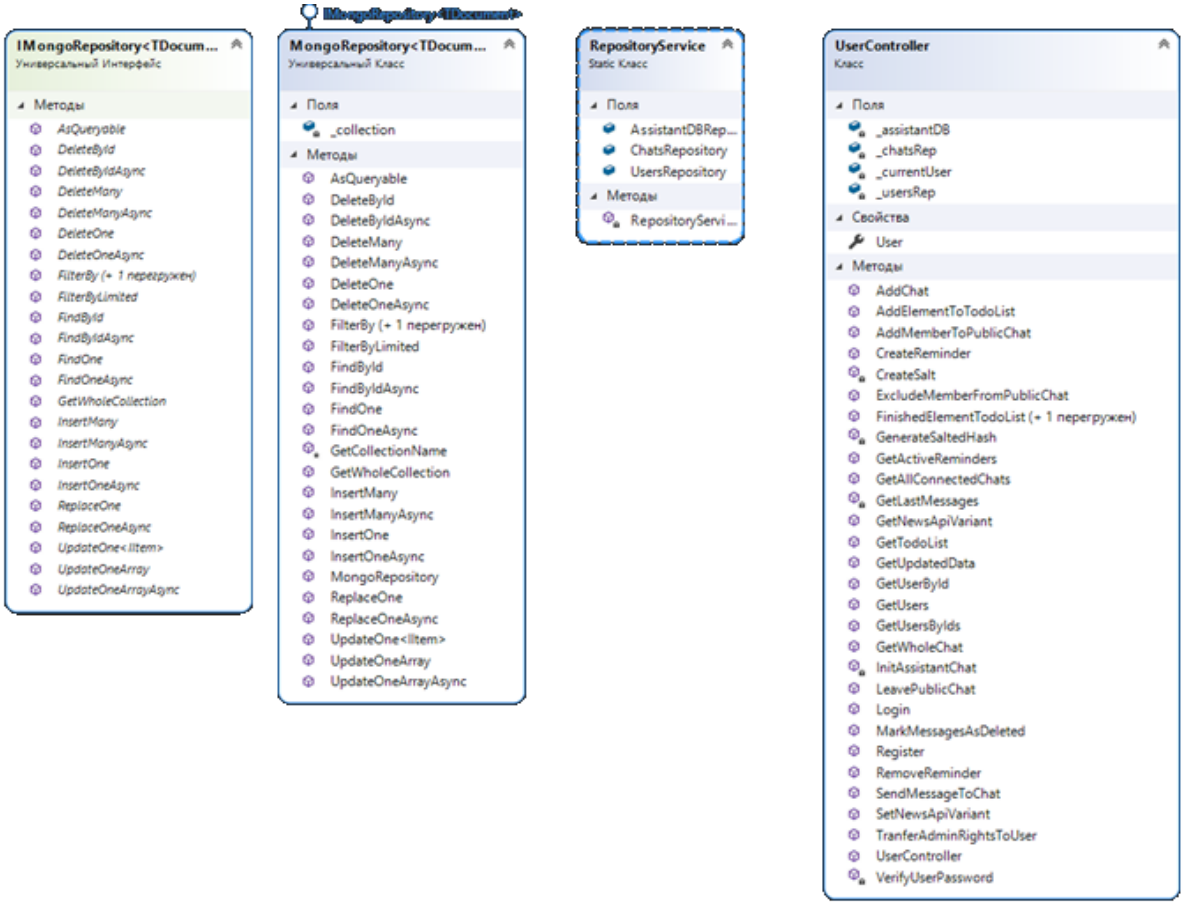


Рисунок 4.4 – Діаграми класів DatabaseModule.cs

Як було описано в теоретичні частині для реалізації прошарку роботи з базою даних був обраний шаблон репозиторію. IMongoRepository.cs – інтерфейс в якому визначені всі CRUD операції та також додаткові необхідні операції для взаємодії з БД. MongoRepository.cs – це реалізація цього інтерфейсу. Передача цього об’єкту по інтерфейсу покращує гнучкість коду та дозволяє використовувати для тестування Mock-об’єкти.

UserController.cs – використовує класи доступу до бази даних щоб реалізувати конкретні операції пов’язані з одним користувачем.

4.2 Проектування та реалізація віртуального асистенту

4.2.1 Загальний опис

Мета віртуального асистента у додатку – забезпечити просте і зручне керування набором функцій. До функцій входять: отримання новин, отримання погоди в певному місті, встановлення нагадувань, ведення списку справ користувача. Враховуючи відносну простоту задачі було вирішено використати метод співставлення тексту з заздалегідь написаними шаблонами. В разі співпадіння з певним шаблоном асистент повинен звернутися до відповідного модуля та виконати потрібні команди.

4.2.2 Роль API

Такий функціонал як отримання новин, отримання погоди потребує зовнішніх джерел інформації. Для цих задач існує безліч безкоштовних ресурсів які дозволяють отримати дані по протоколу HTTP або HTTPS.

Новини – те з чим людина стикається кожен день. Часто люди обирають для себе ресурси з яких хочуть отримувати інформацію. Через це було вирішено надати користувачу право вибору джерела новин. В додатку наявні такі джерела як: New York Times[12], Free News[13]. З огляду на майбутнє розширення в додатку передбачено поповнення цих джерел та визначений узагальнений інтерфейс таких модулів.

Для модуля погоди було обране Open Weather API[14]. Його безкоштовна версія дозволяє робити до 60 запитів за хвилину, що більш ніж достатньо в рамках проекту. API приймає назву міста яке повідомить користувач та видає короткий звіт про погоду.

4.2.3 Нагадування та список справ

Асистент може встановлювати нагадування на певний час який вказав користувач. Основна вимога до таких команд це початок зі слів “Remind to”, “Remind me to” або просто “Remind”. Після цих слів іде вміст нагадування. Користувач може вказати час у цьому ж реченні або зробити це поетапно за

вказівками асистенту. Приклади в одному реченні: “Remind to fix bugs today at 15:35”, “Remind me to do something in 1 minute”. При не вказаному часі асистент перепитає на який час зробити нагадування. Формати часу задані регулярними виразами, розпізнаються такі формати: “in 1 minute” де замість хвилини можна вказати будь-яку одиницю виміру часу (дні, години, неділі), “today at 17:30” де замість today можливо і tomorrow, “17.06.2021 at 14:35” точні дати. При некоректних датах, або минулих датах асистент відправить відповідні пояснення.

Список справ користувача – це задачі які користувач хоче запам’ятати і завершити їх може тільки сам користувач. Робота з цією структурою відбувається за допомогою ключового слова todo. Вказавши одне слово todo асистент відправить наявний список задач, якщо вказати після todo якусь інформацію – задача додається у список. Завершити задачу – тобто видалити її можна за допомогою слів “todo remove”, “todo finish”, “todo end”, “todo delete” і після цього вказати саму задачу або її номер у списку.

4.2.4 Реалізація асистенту

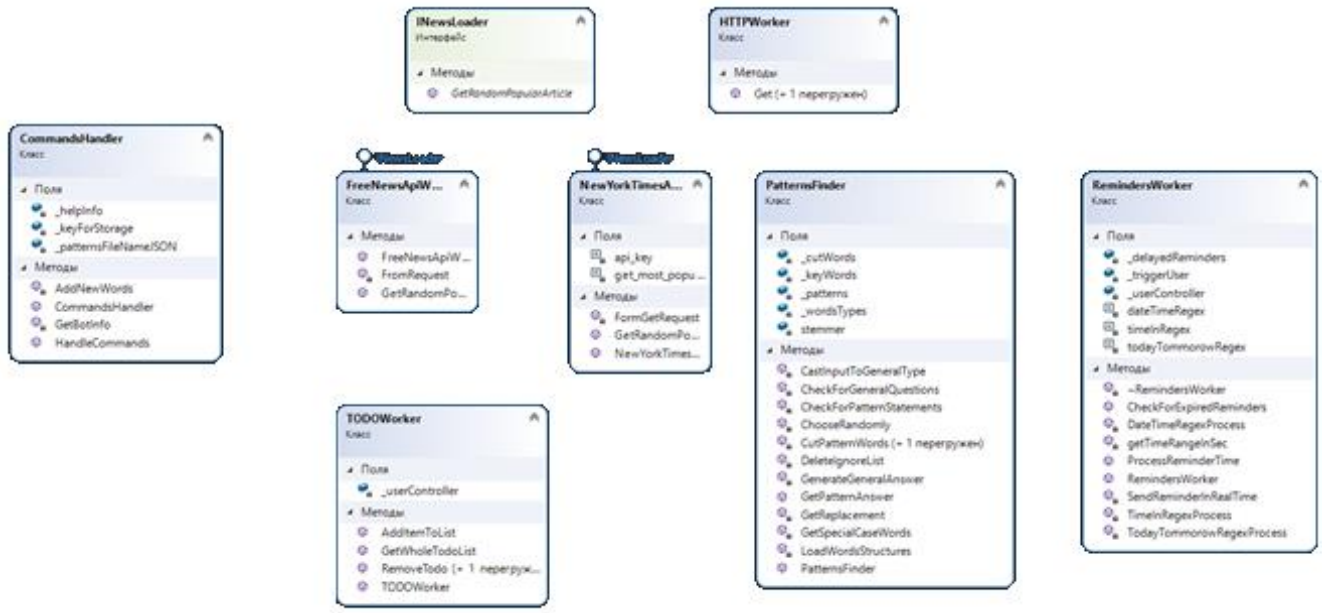


Рисунок 4.5 - Діаграми класів VirtualAssistant.cs

На клієнті асистент – звичайний приватний чат з відповідною іконкою і також цей чат завжди закріплений вгорі всіх чатів. На сервері існує модуль який приймає повідомлення відправлені в чат асистенту та генерує на них відповідь.

В базі даних чат з асистентом зберігається як звичайний приватний чат, але з одним учасником. Документ цього чату створюється при успішній реєстрації користувача. Також в документі користувача є поле “AssistantChatId”, яке вказує до якого чату з асистентом прив’язаний користувач.

Як відбувається розпізнавання шаблонів? В модулі асистенту на сервері є json файл в якому містяться шаблони та деякі знання асистенту. Приклад шаблону для звичайного діалогу (рядки з json файлу):

"* (hello|hi|hey)": "Hello!|hi|Greetings|Hey)",

">p (it) >v (is) *": "Is it really * ?",

В даному випадку ключ це шаблон який можливо буде співпадати з тим що написав користувач. Значення – це відповідь асистента. Написаний парсер (PatternsFinder.cs) який зчитує всі дані з json файлу та при отриманні повідомлень намагається знайти співпадіння.

Маркери в шаблонах:

- “>v” - означає що на цьому місці стоїть дієслово(всі дієслова вказані в файлі)
- “>p” - означає що на цьому місці стоїть займенник(всі займенники вказані в файлі)
- “*” - означає що на цьому місці стоїть будь-який іменник(їх асистент звичайно не знає)
- “>a” - означає що на цьому місці стоїть артикль(всі артиклі вказані в файлі)
- “>s” - означає що на цьому місці стоїть спеціальне питальне слово(всі спец питальні слова вказані в файлі (what, who, why і т.д.))

- “>r” - замінник – означає що після цього маркеру іде слово яке треба замінити через різні роди слів(тобто am замінити на are, you на I) (всі слова-замінники вказані в файлі)

Також після кожного маркеру через пробіл в круглих дужках можна вказати які саме слова треба очікувати. Наприклад:

">p (he|she) >v (can|could|should|may|might|must|will) >v": "What else >v >p do ?",

Тут після займенника в дужках вказано що цими займенниками можуть бути he або she (і тільки вони). Також після маркеру дієслова в дужках вказані всі модальні дієслова.

Даний шаблон розрахований на те що користувач напиши що він щось може/вміє/повинен зробити. Асистент перепитає на це що користувач ще може/вміє/повинен зробити. Це спрямоване для того щоб підтримати діалог при якому асистент чіпляється за ключові слова, щось перепитує, дає відповіді на питання. Паралельні риски у відповіді асистента(як в першому прикладі з привітанням) означають що асистент вибере будь-яку відповідь з розділених рисками навмання.

Є шаблони відповіддю на які будуть конкретні дії. Наприклад:

">v (tell|show) * (some) * (new|news)": "/popular_news",

Як бачимо – відповідь це команда яка починається з символу “/”. Тобто розпізнавши таку команду асистент задіє окремі модулі для цього. В даному випадку буде задіяний модуль для роботи з New York Times API. Асистент отримає з сайту популярну новину навмання, посилання на картинку, посилання на саму новину. Після цього відправить ці дані користувачу.

При питанні про погоду або нагадування активується декілька етапів. Тобто після питання про погоду асистент запитає в якому місті знаходиться користувач. При встановленні нагадування асистент перепитає на який час його встановити.

Нагадування починаються з фраз “Remind me to …” “Remind to ...” Далі асистент запитує дату на яку потрібно назначити нагадування. Приклади які сприймаються: “today at 22:55”, “tomorrow at 20:35”, “in 2 minutes”, “in 5 days”, “05.10.2021 at 22:01”. Також час можна вказати і в одному реченні з самим нагадуванням.

Є команди які може відправити сам користувач:

/help – Описує яким чином користуватись ботом.

/info – Короткий звіт про знання асистенту.

4.3 Схема бази даних

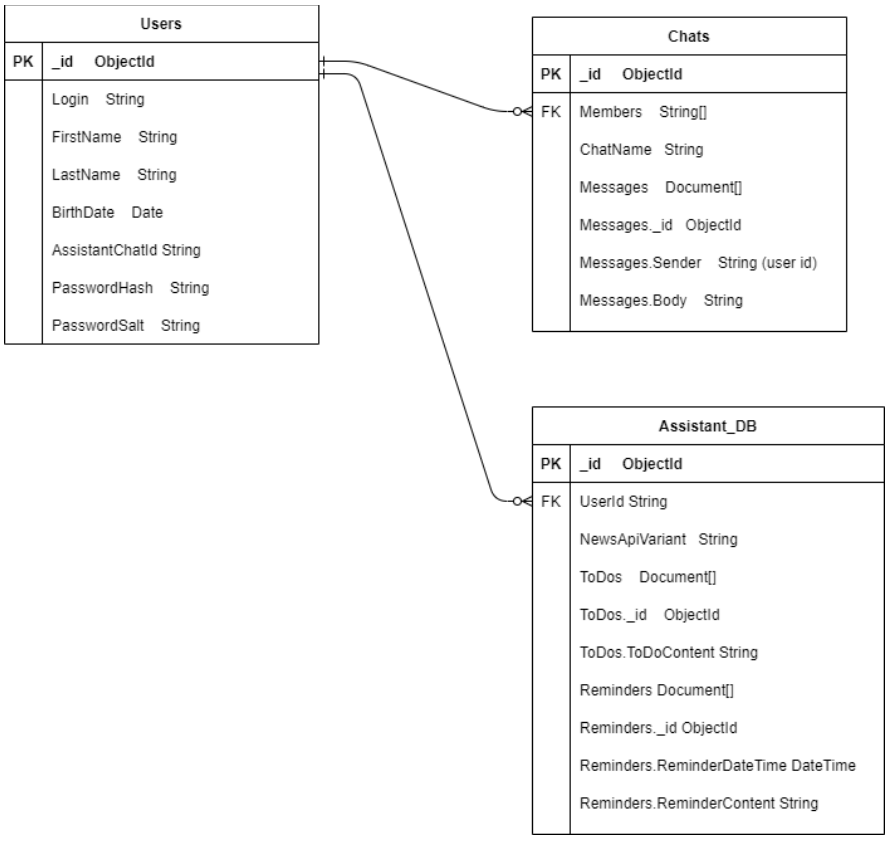


Рисунок 4.6 – Er модель бази даних

Про користувача зберігаються такі дані як: унікальний логін, ім’я, прізвище, дата народження, ідентифікатор чату з асистентом, хеш паролю, сіль паролю. В документах чатів зберігаються список учасників у вигляді масиву ідентифікаторів,

назва чату та масив повідомлень. Також для кожного користувача заводиться документ з даними які він відправляє асистенту. Це джерело інформації для новин, список справ, список нагадувань, також стрічка з ідентифікатором користувача з якому ці дані належать. Через те що була обрана база даних за парадигмою NoSql то реляційна модель відсутня.

4.4 Реалізація мобільного додатку

4.4.1 Навігація по додатку

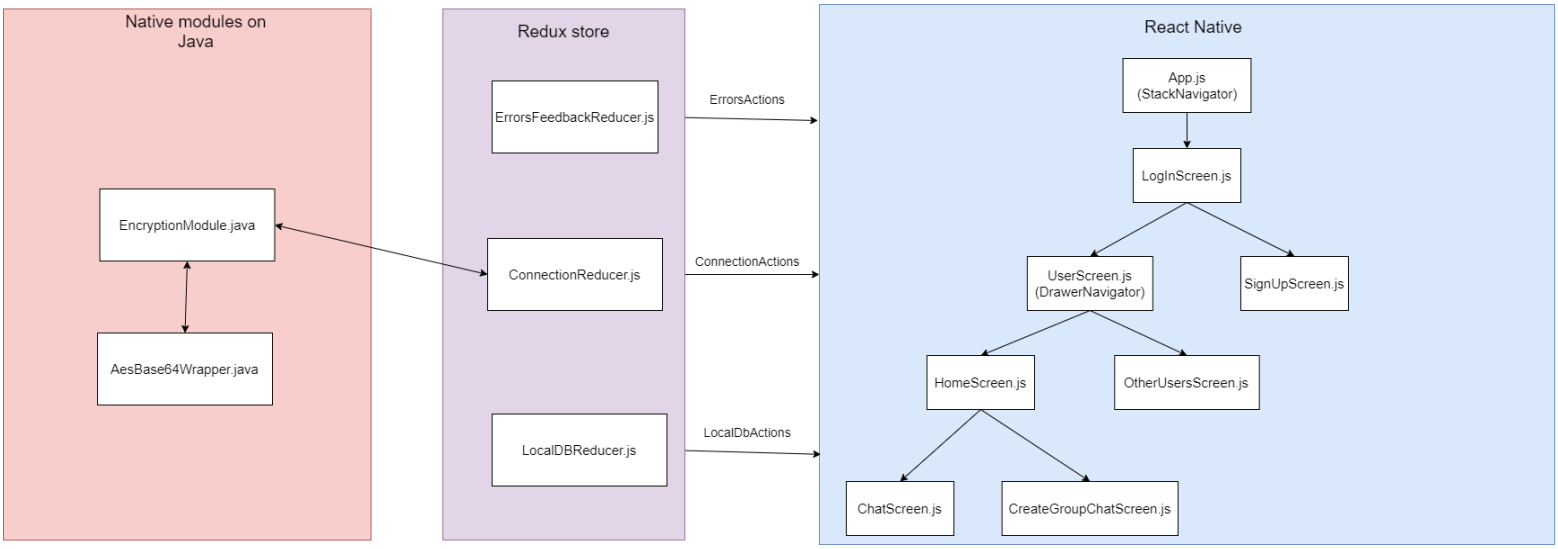


Рисунок 4.7 - Схема екранів та їх взаємодія з Redux сховищем

Важливим елементом в мобільних додатках є зручна навігація по сторінкам з контентом. Для цього було об’єднано декілька навігаторів.

StackNavigator, який працює з екранами як зі стеком – екрани можна додавати на стек або видаляти тим самим повертаючись до попереднього елементу.

DrawerNavigator дозволяє працювати з екранами як з таблицею. Переміщатись по ним дозволяє бокова панель яку можна потягнути або закрити. Така панель є дуже типовою для такого роду додатків.

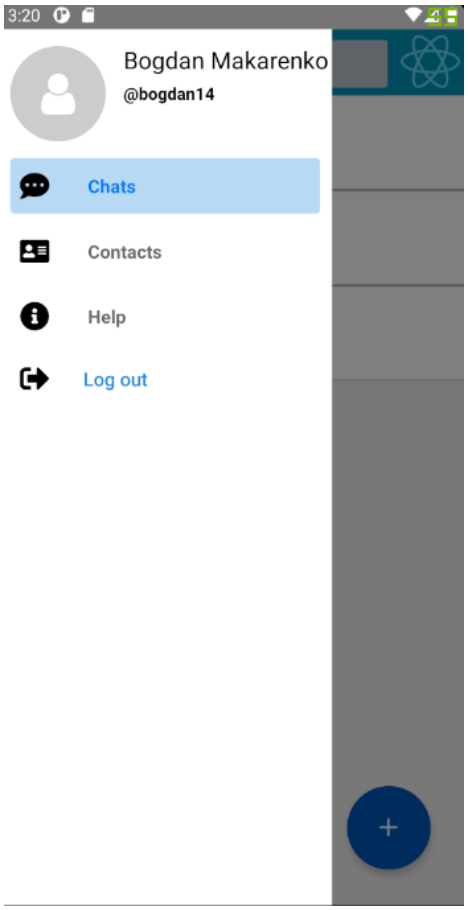


Рисунок 4.8 – Бічна панель

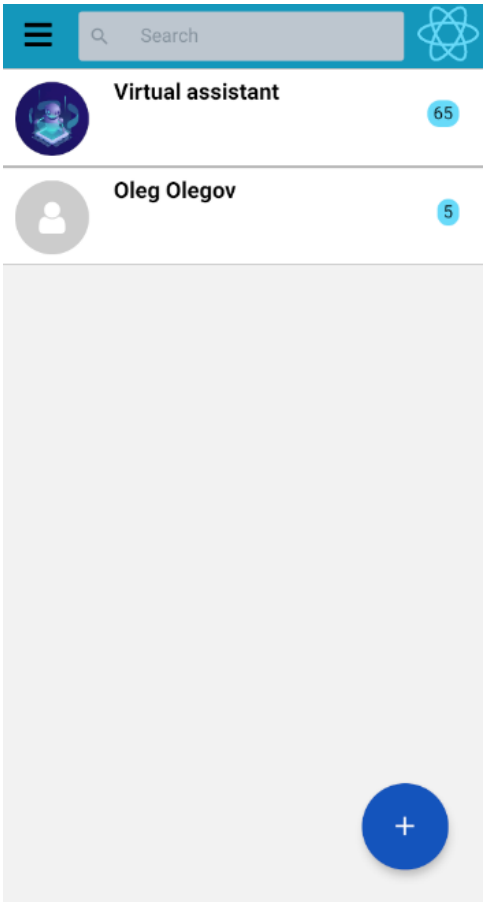


Рисунок 4.9 – Панель закрита

4.4.2 Реєстрація та аутентифікація користувачів

LogInScreen та SignUpScreen відповідають за вхід та реєстрацію користувачів. При наявності помилок з боку користувача виводяться відповідні повідомлення. Також екрани адаптивні і при малих розмірах екрану з’являється вертикальна прокрутка.

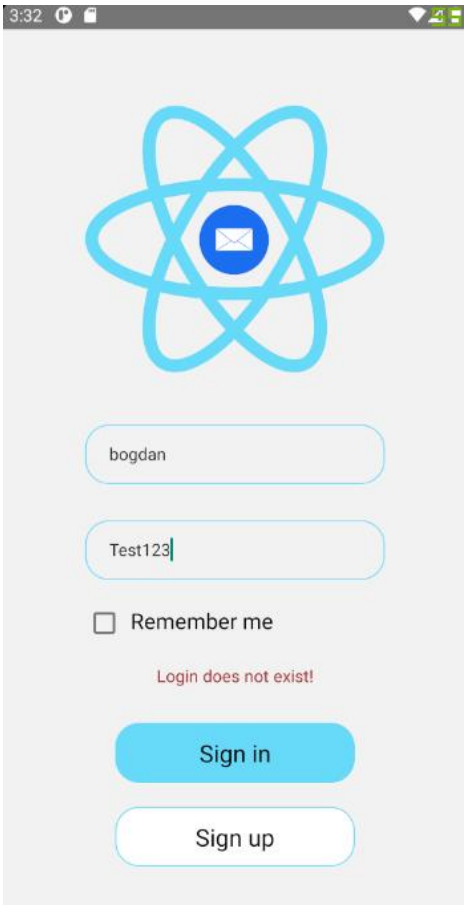


Рисунок 4.10 – невірний логін

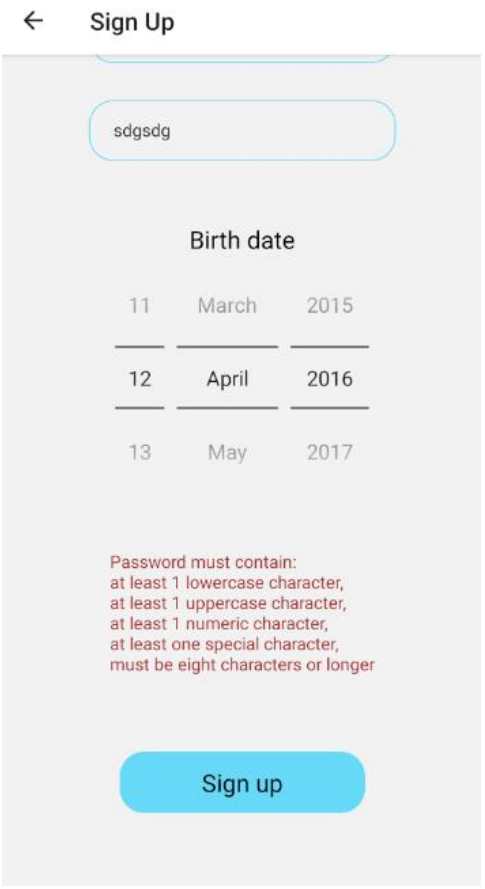


Рисунок 4.11 – пароль не відповідає вимогам безпеки

4.4.3 Пошук

Глобальний пошук дозволяє користувачу шукати інших людей прямо з головного екрану. Такий підхід використовується в сучасних месенджерах на зразок Telegram. Крім того пошук відбувається одразу і по наявним чатам. На рисунку 4.12 зображений приклад глобального пошуку. Також на екрані з контактами можна проводити пошук по людям (рисунок 4.13). Таким же чином проводиться пошук під час створення групових чатів.

Пошук у застосунку працює таким чином: коли користувач змінює поле вводу запускається таймер на 1.5 секунди. Якщо користувач щось змінив протягом цього часу то таймер знову виставляється на 1.5 секунди. Якщо час збігає, тобто нічого не змінювалось протягом цього часу то запит відправляється на сервер.

Такий підхід дозволяє зменшити частоту запитів, що в свою чергу знімає навантаження.

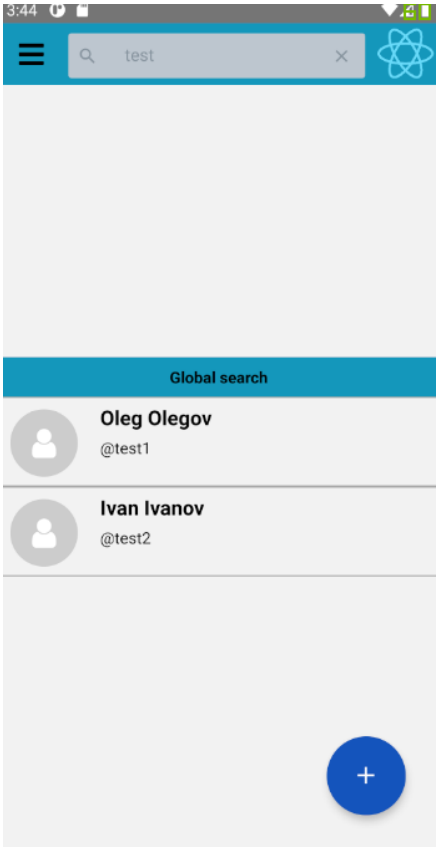


Рисунок 4.12 – глобальний пошук

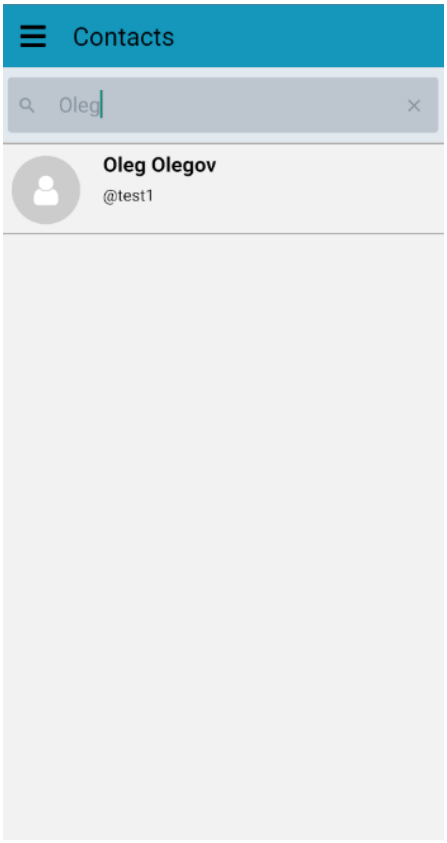


Рисунок 4.13 – пошук по контактам

4.4.4 Приватні чати

Щоб створити приватний чат з людиною достатньо знайти її в глобальному пошуку або в пошуку контактів та натиснути на неї. При відправці першого повідомлення чат ініціалізується та з’явиться в іншого користувача. Відправляти можна текстові повідомлення, посилання, посилання на картинки. Також можна вибрати певні повідомлення та видалити їх для себе назавжди.

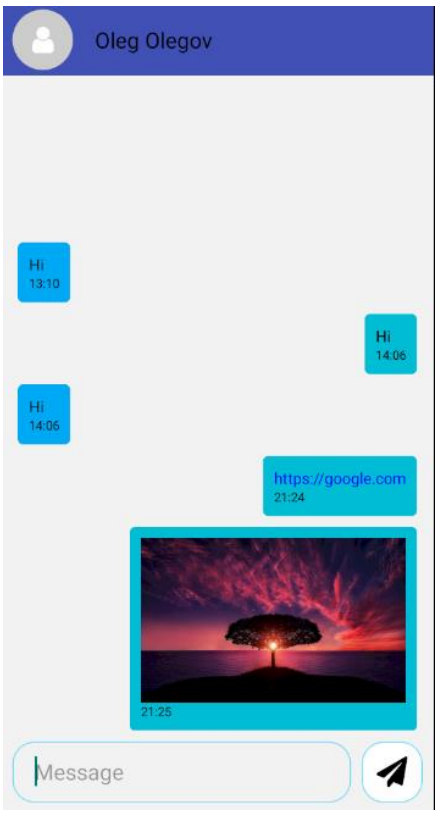


Рисунок 4.14 – приватний чат

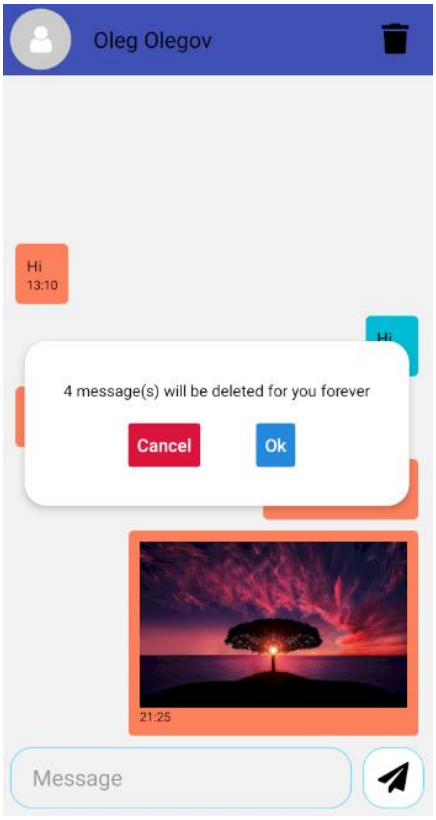


Рисунок 4.15 – видалення повідомлень

4.4.5 Групові чати

Щоб створити груповий чат потрібно натиснути на синю плаваючу кнопку зі знаком плюс (рисунок 4.12). Далі необхідно ввести назву чату та додати хоча б двох інших користувачів.

Користувач який створив чат є адміністратором та має право додавати або видаляти користувачів. Також він може передати свої права іншому користувачу. Звичайний користувач може тільки покинути чат.

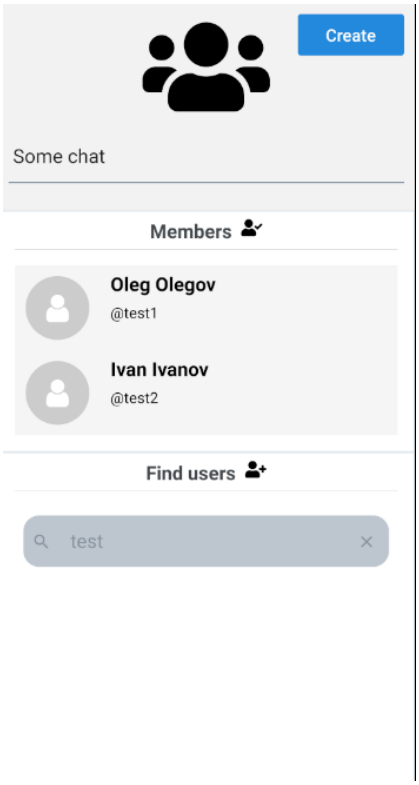


Рисунок 4.16 – процес створення групового чату

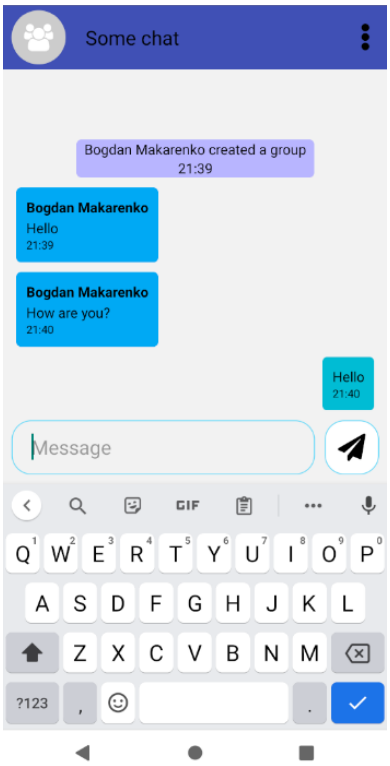


Рисунок 4.17 – вигляд групового чату

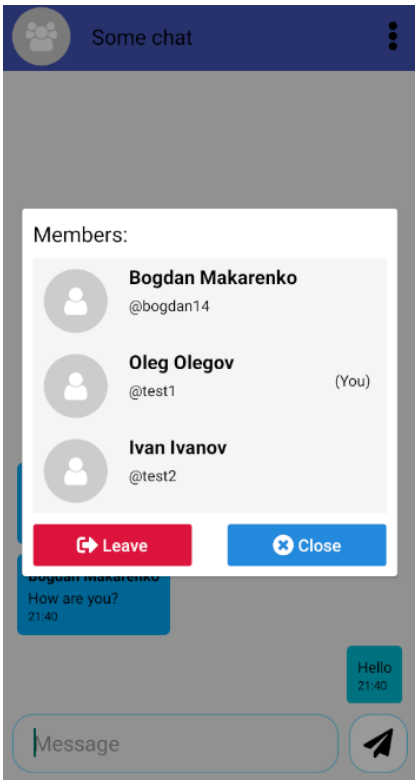


Рисунок 4.17 – панель звичайного користувача

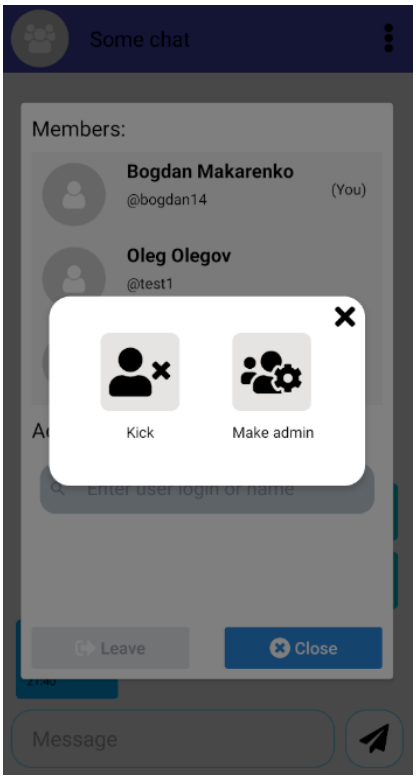


Рисунок 4.18 – панель адміністратора чату

4.4.6 Чат з віртуальним асистентом

Цей чат завжди знаходиться згори всіх чатів та має спеціальну іконку (рисунок 4.9)

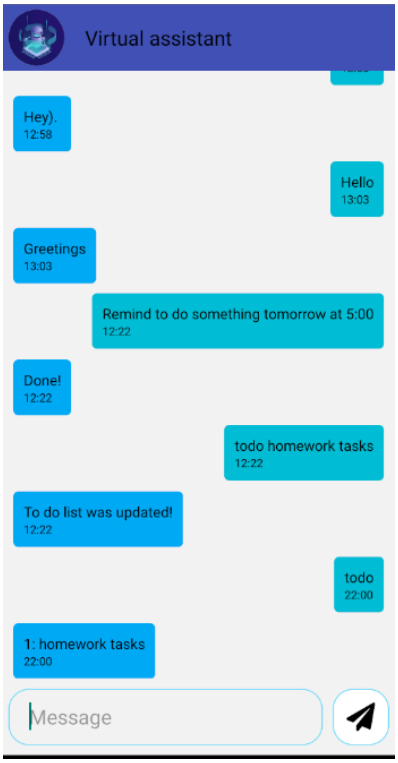


Рисунок 4.19 – нагадування та todo

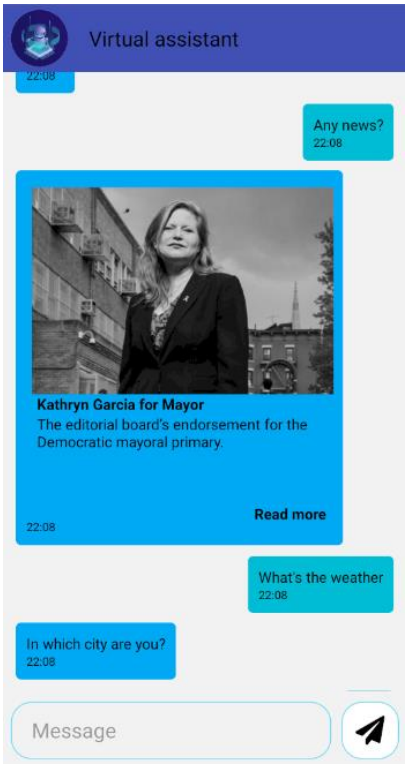


Рисунок 4.20 – перегляд новин

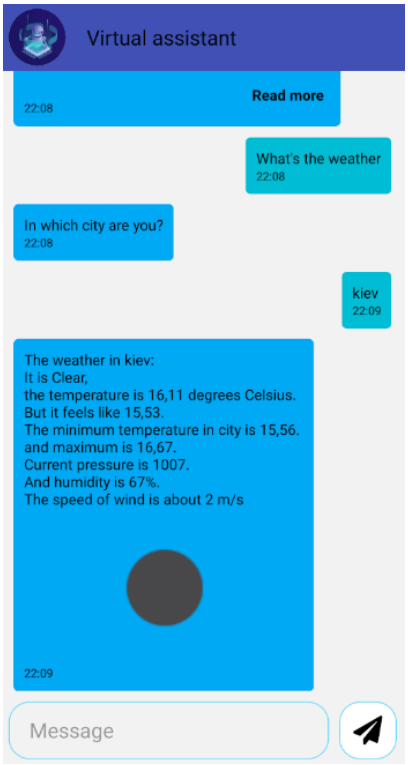


Рисунок 4.21 – відображення погоди

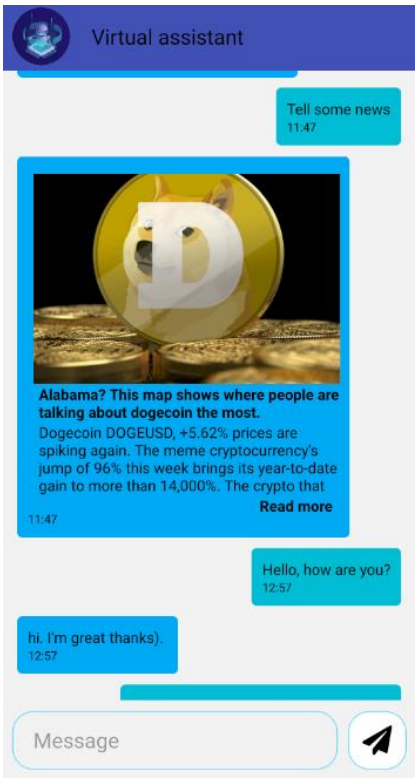


Рисунок 4.22 – інший ресурс новин

4.5 Висновки до розділу 4

В даному розділі було розглянуто реалізацію додатку. Приведені схеми архітектури та бази даних застосунку. Описана робота з клієнтським додатком та наведені скріншоти з нього. Описаний принцип роботи, реалізація та можливості віртуального асистенту.

Висновки

Розробка даного додатку допомогла краще зрозуміти технології які використовуються в сучасних серверних застосунках та програмах для мобільних пристроїв. А саме отриманий досвід роботи з такими технологіями як C# .Net Core, React Native, Redux, Javascript, Java , нереляційними базами даних MongoDB.

При розробці віртуального асистенту були проаналізовані розповсюджені API для отримання таких даних як новини, погода. Також був реалізований метод розпізнання повідомлень користувача за допомогою шаблонів, що в свою чергу допомогло зробити додаток простим та зручним у використанні.

З подальшого розширення додатку можна виділити такі кроки:

- Використання алгоритмів штучного інтелекту замість шаблонів для реалізації розумного асистенту
- Реєстрація в системі за допомогою інших сервісів таких як Google.
- Реалізація push-повідомлень та збільшення користі нагадувань, повідомлень в реальному часі
- Розширення налаштувань для збільшення гнучкості та функціональності. Наприклад: зміна даних користувача, зміна паролю, інтеграція з системними контактами.
- Підтримка відправки файлів, аудіо, відео, голосових повідомлень, геолокацій.

Список використаних джерел

1. Опис можливостей продукту Facebook M та причинах його закриття [Електронний ресурс] / Кейзі Ньютон // theverge.com – 2018. - Режим доступу до ресурсу: https://www.theverge.com/2018/1/8/16856654/facebook-m-shutdown-bots-ai
2. Багаторівнева архітектура [Електронний ресурс] / Марк Річардз //taithienbo.com
3. Комп'ютерні мережі: [навчальний посібник] / А. Г. Микитишин, М. М. Митник, П. Д. Стухляк, В. В. Пасічник. — Львів: «Магнолія 2006», 2013. — 17 с.
4. Шаблон – пул потоків [Електронний ресурс] // medium.com – 2016. - Режим доступу до ресурсу: https://medium.com/@dholnessii/the-thread-pool-pattern-7227eb9ec2b6
5. JSON проти XML [Електронний ресурс]/ Рамай Шанкар // hackr.io – 2021. - Режим доступу до ресурсу: https://hackr.io/blog/json-vs-xml
6. Симетричні та асиметричні алгоритми шифрування [Електронний ресурс]// rapidsslonline.com – 2021. - Режим доступу до ресурсу: https://www.rapidsslonline.com/blog/fundamental-differences-between-symmetric-and-asymmetric-encryption/
7. SQL та NoSQL підходи [Електронний ресурс]/ Бенджамін Андерсен// hackr.io – 2020. - Режим доступу до ресурсу: https://www.ibm.com/cloud/blog/sql-vs-nosql
8. Можливості MongoDB [Електронний ресурс]// mongodb.com. - Режим доступу до ресурсу: https://www.mongodb.com/why-use-mongodb
9. Dao та Repository на прикладі Java [Електронний ресурс]/ Аншул Бансал// baeldung.com – 2020. - Режим доступу до ресурсу: https://www.baeldung.com/java-dao-vs-repository
10. Компоненти у React Native [Електронний ресурс]// reactnative.dev. - Режим доступу до ресурсу: https://reactnative.dev/docs/components-and-apis

11. Компоненти розширення з бібліотеки React Native Elements // reactnativeelements.com - Режим доступу до ресурсу: https://reactnativeelements.com/docs/overview
12. Опис New York Times API // developer.nytimes.com - Режим доступу до ресурсу: https://developer.nytimes.com/docs/most-popular-product/1/overview
13. Опис Free News API // newsapi.org - Режим доступу до ресурсу: https://newsapi.org/docs
14. Опис Open Weather API // openweathermap.org - Режим доступу до ресурсу: https://openweathermap.org/current

Додатки

Додаток А

Sequence діаграма оновлення чатів в реальному часі на прикладі двох користувачів.