КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

Розробка телеграм бота «Електронний помічник старости»

Кваліфікаційна робота бакалавра
студента 4 року навчання
спеціальність: 123 «Комп'ютерна інженерія»
Богдана СПІВАКА

Науковий керівник:
канд. фіз.-мат. наук, доцент
Михайло КОНОНОВ

Рецензент:
канд. фіз.-мат. наук Олександр СУДАКОВ,
доцент кафедри медичної фізики

До захисту допускаю

Завідувач кафедрою
Юрій БОЙКО

Ухвалено на засіданні кафедри "____"_____2022 р., протокол № ___

КИЇВ - 2022

РЕФЕРАТ

Обсяг роботи 46 сторінок, 36 ілюстрацій,1 таблиця, 15 джерел посилань.

МЕСЕНДЖЕР, ТЕЛЕГРАМ, API , PYTHON, SQLALCHEMY , POSTGRESQL

Актуальність роботи полягає в тому, що наразі майже всю інформацію щодо навчання студенти отримують в месенджерах , але не завжди ця інформація структурована. Вирішити цю проблему можна за допомогою розробки ботів. Також актуальність роботи полягає в дослідженні етапів та засобів розробки бота.

Головною метою роботи є розробка боту, який дасть змогу полегшити отримання та розповсюдження інформації щодо навчання.

ЗМІСТ

Скорочення та умовні позначення

API – прикладний програмний інтерфейс

JSON – текстовий формат обміну даними оснований на JavaScript

бд – база даних

SQL — мова програмування для взаємодії з базами даних

Месенджер – додаток створений для спілкування користувачів

ВСТУП

Зараз важко уявити наше життя без інтеренту, він дав нам змогу швидко отримувати актуальні новини, знаходити потрібну інформацію , спілкуватися з іншими людьми, швидко знаходити вирішення проблеми та багато іншого. Майже для кожної потреби сучасного користувача існують відповідні застосунки.

Найрозповсюдженішим інструментом для спілкування вважаються месенджери, які поєднують у собі весь необхідний для комфортного життя сучасної людини функціонал. За їх допомогою можна переглядати новини, для цього достатньо підписатися на канал який нас цікавить, знайомитися та спілкуватися з людьми, це можна робити як в публічних бесідах так і приватних повідомленнях, здійснювати покупки та навіть використовувати як пошукову систему. Звісно важко описати весь доступний функціонал, але більшість функцій не було би можливим реалізувати без чат ботів. Вони дозволяють реалізовувати найрізноманітніші функції поверх інтерфейсу месенджеру починаючи від простих розсилок закінчуючи іграми.

Вже не один семестр навчання проходить в дистанційному режимі і інформування студентів суттєво ускладнилося. Цю проблему вирішило спілкування у месенджерах, але іноді доволі важко знайти потрібну інформацію, тому виникає потреба в її структуризації. Одним з варіантів вирішення цієї проблеми є створення спеціальних інформаційних каналів, але цей спосіб має свої недоліки. Головний з них це те що все знаходиться докупи: розклад, інформація з різних предметів, інша інформація з приводу навчання, завдання, тощо

В даній роботі була поставлена задача вивчення можливостей чат ботів, вибір інструментів їх розробки та створення власного продукту

«Електронний помічник старости», який дасть змогу спростити доступ до інформації та її розповсюдження. Цей продукт може використовувати будь який студент для більш зручного доступу до навчальної інформації.

1.Боти

З розвитком технологій зникає необхідність у виконанні повторюваних задач людиною, бо програма може зробити це дешевше , швидше, точніше та без зайвих запитань.

Бот- скорочено від «робот» - це програма, яка виконує запрограмовані задачі, які імітують діяльність людини. Боти працюють згідно заздалегідь прописаного людиною алгоритму. Зазвичай вони використовуються для автоматизації конкретних задач, тобто вони можуть допомогти підприємцю або якійсь організації зекономити час та гроші, виконуючи роботу замість співробітника.

Існує багато типів ботів, деякі з них[4]:

• Чат-боти – для імітації розмови з людиною.

• Соціальні боти – боти на платформах соціальних мереж.

• Shopbot – бот який моніторить товар, який відповідає певним критеріям, та робить автоматично покупку за вас.

• Ноуботи – збирає інформацію, яка відповідає певним критеріям на різних веб ресурсах.

• Сканери – для збору змісту певних сайтів та подальшої їх індексації.

• Моніторинг-боти – для моніторингу роботоздатності певної системи.

•Месенджер боти – боти на платформах месенджерів.

Для реалізації застосунка було вирішено розробляти-чат бота у месенджері. Цей вибір був зроблений через ряд переваг:

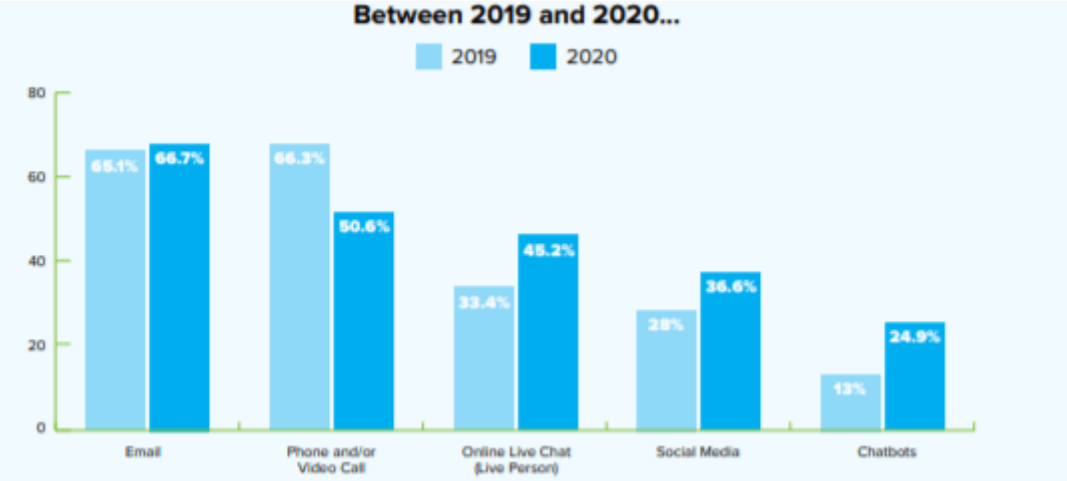
а) Користувачу не потрібно додатково щось встановлювати, адже він може вирішувати свої задачі не виходячи з улюбленого месенджеру

б)Не потрібно витрачати час на розробку інтерфейсу користувача

в)Нативність інтерфейсу, за рахунок того що користувач вже знайомий з месенджером

1.1Актуальність ботів

Згідно звіту Drift State of Conversational Marketing[11] у 2020 році використання ботів для комунікації з брендом виросло вдвічі порівняно з 2019, діаграма проілюстрована на рис. 1.1. Це говорить про стрімкий розвиток та розповсюдження цього способу комунікації.



«Рисунок 1.1 - Статистика способів комунікації з брендами»[11]

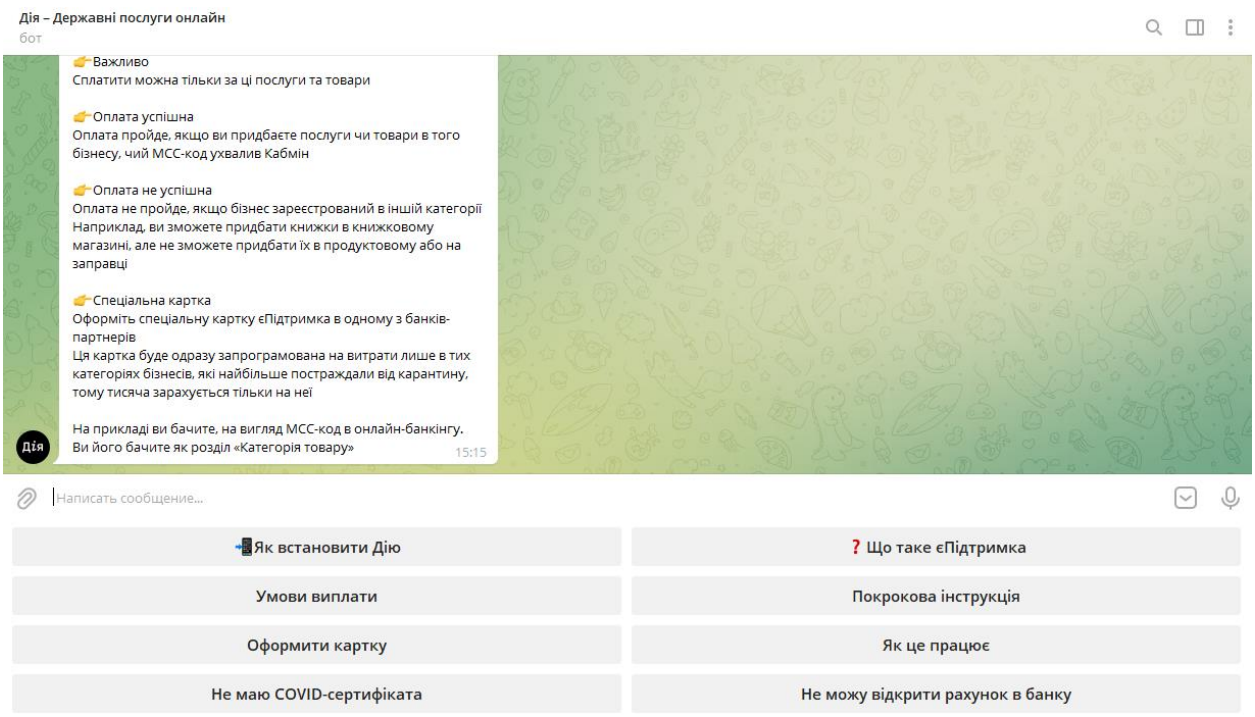
За даними опитування Київського міжнародного інституту соціології[12], які продемонстровані у таблиці 1.1, за вересень 2021 року найбільш популярними месенджерами є Viber, яким користується 73,6% опитаних, Facebook Messenger – 42,7%, Telegram — 31,6% , WhatsApp — 25,3%.

«Таблиця 1.1 Найпопулярніші месенджери в Україні станом на вересень 2021 року»

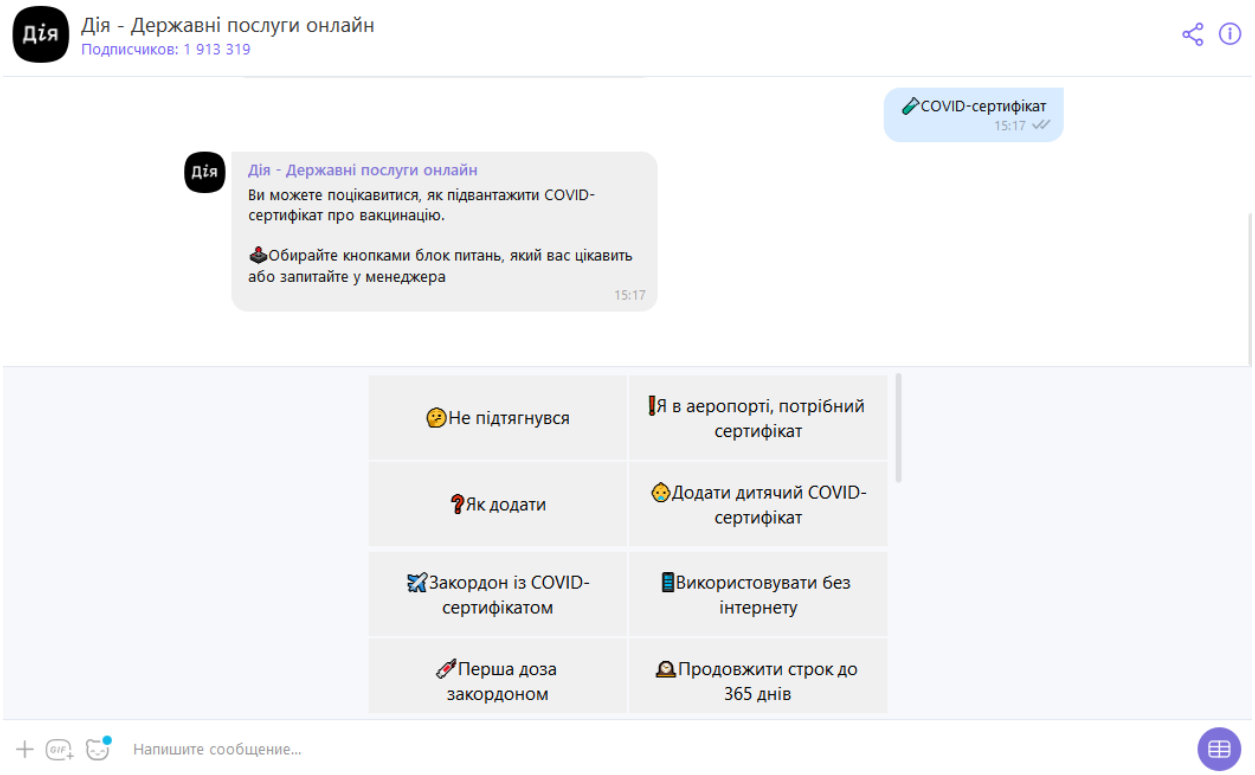
Мобільний застосунок	**%**
Viber(вайбер)	73,6
WhatsApp(вотсап)	25,3
Месенджер Фейсбуку	42,7
Telegram(Телеграм)	31,6
Twitter	4,9
Signal	3,8
Skype	11,3
Instagram	26,4
TikTok	9,1
Інше	2,6
Не користуюсь	18,7
Важко сказати/Відмова відповідати	0,8

1.2 Огляд месенджерів

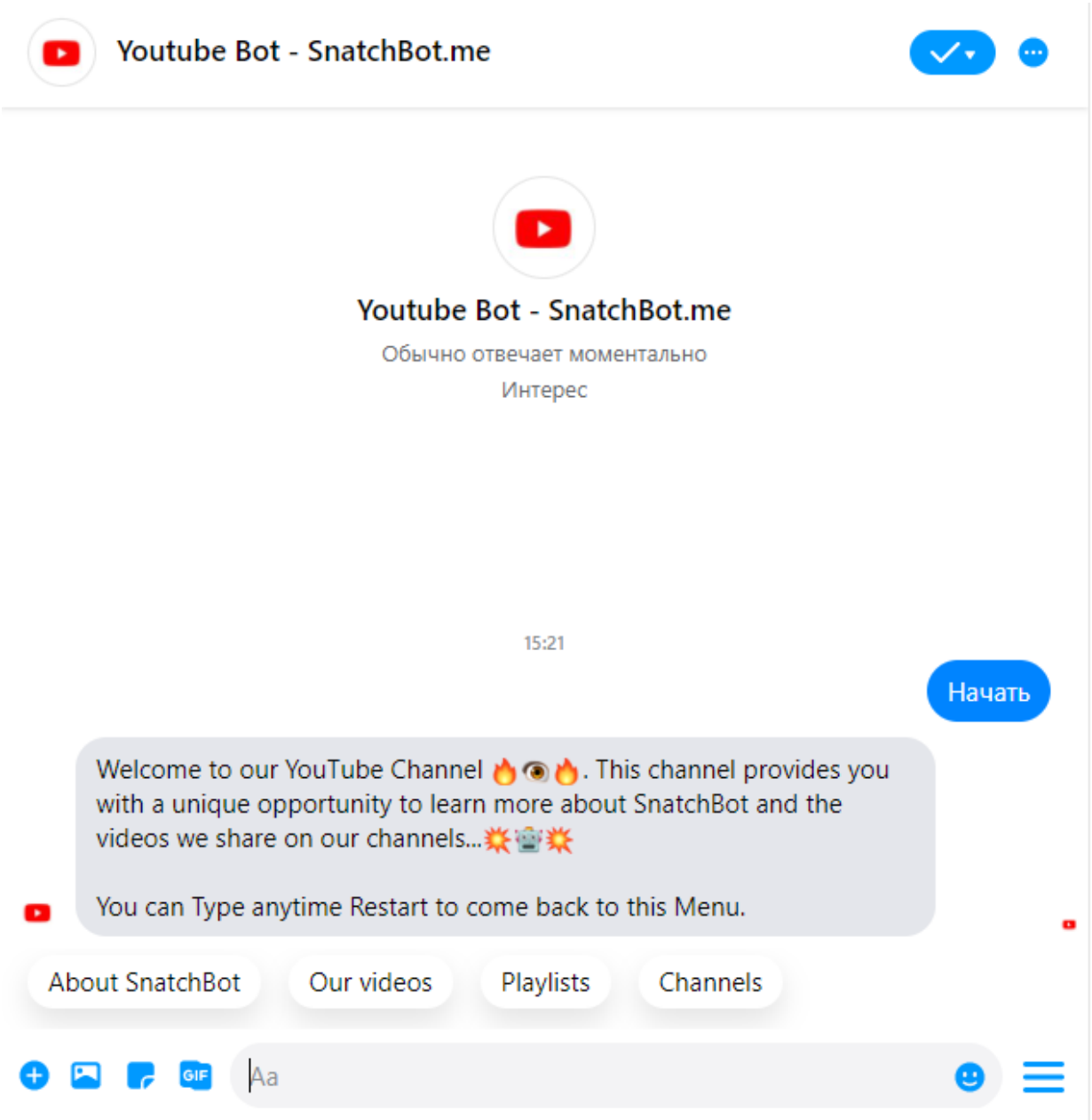
Для огляду я обрав три найпопулярніших месенджера, такі як Viber, Telegram та Facebook Messenger. Інтерфейс ботів та самих додатків майже не відрізняється, але на мою суб’єктивну думку інтерфейс телеграму найприємніший для сприйняття. Цей факт також вплинув на остаточний вибір месенджеру для розробки. На рис. 1.2 зображений приклад телеграм бота ,на рис. 1.3 – приклад бота у вайбері та на рис. 1.4 - приклад бота у фейсбук месенджері.



«Рисунок 1.2 - Приклад телеграм бота»



«Рисунок 1.3 - Приклад бота у вайбері»



«Рисунок 1.4 - Приклад бота у фейсбук месенджері»

Звісно, при розробці бота у месенджері необхідно враховувати популярність даного месенджеру серед тієї аудиторії на яку він розрахований. Наприклад серед студентів факультету радіофізики електроніки та комп'ютерних систем найбільш популярним є саме телеграм, тому для спілкування під час дистанційного навчання ми використовуємо саме цей ресурс.

Отже, підхід до розробки чат боту у месенджері не відрізняється від розробки додатку. Варто розуміти що функціонал може бути ідентичним,

відмінності тільки у способі взаємодії з користувачем. Тому кращим рішенням було обрати розробку чат бота у месенджері. Порівнюючи інтерфейс користувача у різних месенджерах я зрозумів що цей параметр не є ключовим при виборі платформи, тому мій вибір був на користь телеграму через те ,що більшість моїх однокурсників використовують саме його.

2 Огляд засобів розробки

2.1 Python

Для написання телеграм бота можна використовувати будь яку мову програмування яка надає можливість надсилання та обробки HTTP запитів. Деякі з найпопулярніших: Python , Java, PHP та Ruby. В першу чергу звісно потрібно обирати ту мову, з якою більш знайомий, але саме Python є гарним вибором через ряд причин:

- легкий для розуміння та код дуже легко читається, звісно при умові його правильного оформлення

- той самий код ми можемо запустити в Linux, MacOS або Windows

- існує величезна кількість бібліотек та фреймворків Python , завдяки яким кількість строк коду зменшується а його якість та швидкість зростає

-придатний для вирішення задач різної складності

-відмінна документація та величезна кількість форумів з людьми, які готові допомогти вирішити проблему

Для розробки чат бота мовою Python можна використовувати такі бібліотеки:

а)pyTelegramBotAPI (TeleBot) - синхронна бібліотека для створення телеграм ботів.

Саме цю бібліотеку я і обрав.

б)Aiogram - асинхронна бібліотека для полегшення роботи з Telegram Bot Api

Якщо обирати серед цих двох бібліотек, то потрібно звертати увагу на потенційну кількість користувачів боту. Якщо проект та кількість користувачів велика та потенційно буде збільшуватися, то очевидним вибором буде

aiogram , завдяки асинхронності декілька запитів можуть оброблюватися паралельно. В моєму випадку кількість користувачів не буде великою , тому й в асинхронності немає необхідності. Серед переваг бібліотеки TeleBot відмічу відмінну документацію та велику кількість прикладів у вільному доступі. Обидві ці бібліотеки існують як оболонки HTTP- запитів, задля спрощення розробки застосунку.

2.2 PostgreSQL

PostgreSQL - це об'єктно-реляційна база даних з відкритим початковим кодом, яка підтримує реляційні(SQL) та нереляційні(JSON) запити[7]. Вона вважається найактуальнішою та найтехнохнологічнішою системою керування базами даних. Завдяки великій спільноті доступна велика кількість розширень та додатків , які спрощують роботу з цією СУБД.

Серед основних переваг відмічу:

-безкоштовна ліцензія та відкритий початковий код

-надійність, яка була досягнута завдяки спільноті, котра оперативно виправляє будь-які помилки та постійно підвищує продуктивність роботи СУБД

-розширюємість, користувачі можуть створювати свої власні функції, типи даних, тощо

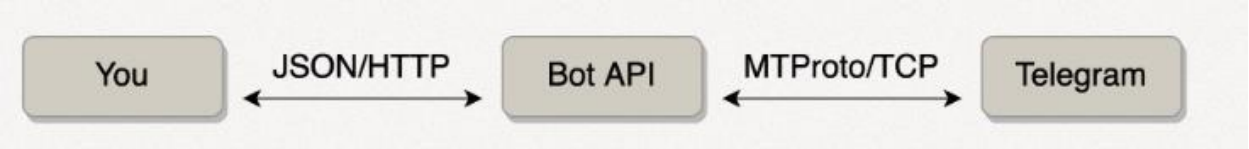
Отже , стабільне вдосконалення , функціональність та універсальність роблять PostgreSQL найкращим вибором, як для простих, так і для складних проектів.

2.3 Telegram Api

Telegram API- це API , через який ваш телеграм-застосунок зв’язується з сервером, та виключає необхідність розбиратися в тому як працює протокол

шифрування[3]. Для шифрування повідомлень телеграм використовує протокол MTProto. Telegram API є відкритим, тому кожен бажаючий може створити свою модифікацію клієнту для месенджера.

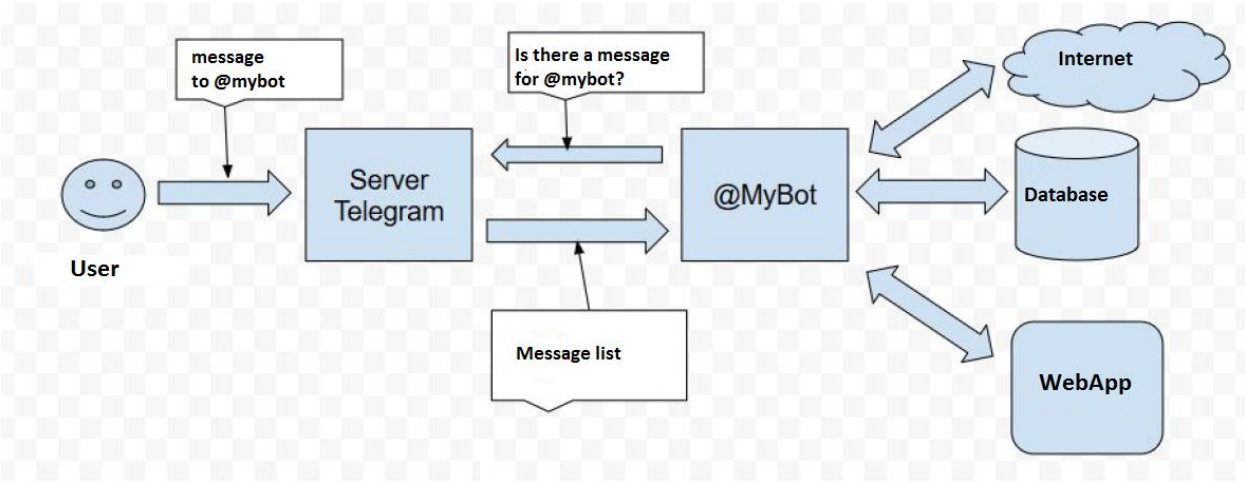
Для написання телеграм ботів був розроблений Telegram Bot API , який являє собою надбудову над Telegram API. В Bot API спрощені робота з вебхуками, розмітка повідомлень, та інше. При необхідності позбавитись від певних обмежень BOT API можна працюючи напряму через Telegram API. Розробники телеграму надали можливість використання локального серверу BOT API, який надає ряд додаткових можливостей таких ,як використання будь якого локального адреса для вебхука, завантаження файлів будь якого розміру та інші. Схема взаємодії з Bot API зображена на рис. 2.1



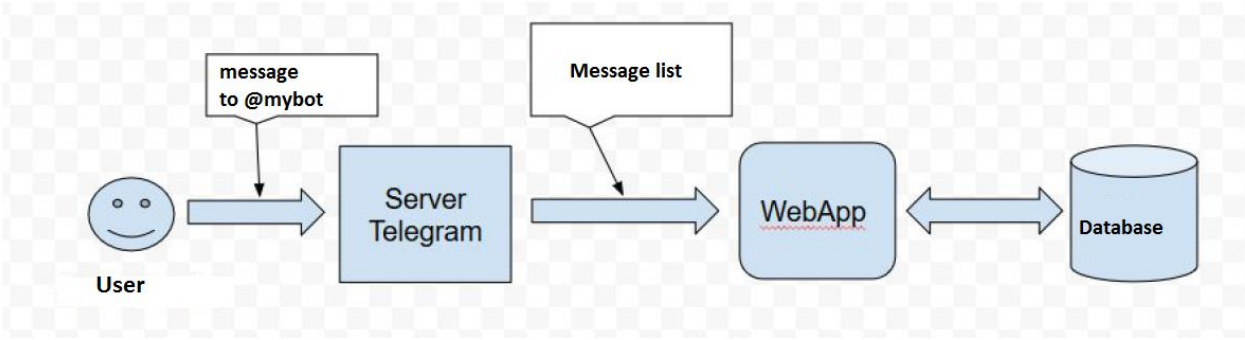
«Рисунок 2.1 - Схема взаємодії з Bot API»[15]

Отримувати повідомлення про дії користувача можна двома способами:
-Polling - періодично опитувати сервер телеграма на наявність оновлень. Схема зображена на рис. 2.2

-*Webhook - телеграм сам відправляє повідомлення, коли користувач взаємодіє з ботом.* Схема зображена на рис. 2.3



«Рисунок 2.2 – Схема взаємодії за допомогою polling»[14]



«Рисунок 2.3 – Схема взаємодії за допомогою Webhook»[14]

У другому випадку необхідно налаштувати сервер, та зробити SSL сертифікат, але в будь якому випадку будуть приходити об’єкти Update.

Об’єкт Update має одне обов’язкове поле update_id, яке є ідентифікатором оновлення, та максимум одне необов’язкове поле, таке як message, channel_post, poll , chat_join_request , тощо.

2.4 SQLAlchemy

Існує два підходи до керування реляційною базою даних:

а)низькорівневий -керування базою даних за допомогою команд SQL

б)високорівневий

Використовується об’єктно реляційне перетворення(ORM - object relational mapper). Тобто ми можемо працювати з базою даних за допомогою об’єктно орієнтованого коду.

SQLAlchemy – це фреймворк для керування базами даних, який може реалізовувати обидва підходи[11]. SQLAlchemy ORM слугує для створення та керування базою даних за допомогою Python класів. Тобто можна керувати базою даних не пишучи SQL скрипти .

Серед основних переваг SQLAlchemy можна відмітити:

-можливість визначення бази даних у коді програми

-синхронізація моделі та схеми, тобто при зміні моделі змінюється і схема бази даних

-можливість заповнення бд даними прямо у коді Python

-відмінна документація

2.5 Docker

Docker - це технологія , яка використовується для розробки та запуску програм в ізольованому середовищі розробки[13]. Для цього використовуються контейнери, які працюють на рівні операційної системи. Контейнер Docker - це автономний виконуючий програмний пакет, який сполучає у собі все необхідне для легкого запуску середовища виконання коду програми[13]. На нього ніяк не впливає операційна система та її налаштування. Порівнюючи з віртуальними машинами контейнери займають в десятки разів менше пам'яті для зберігання, швидше запускаються та не потребують складних налаштувань системи.

Отже, можна відмітити такі переваги:

-можливість резервного копіювання

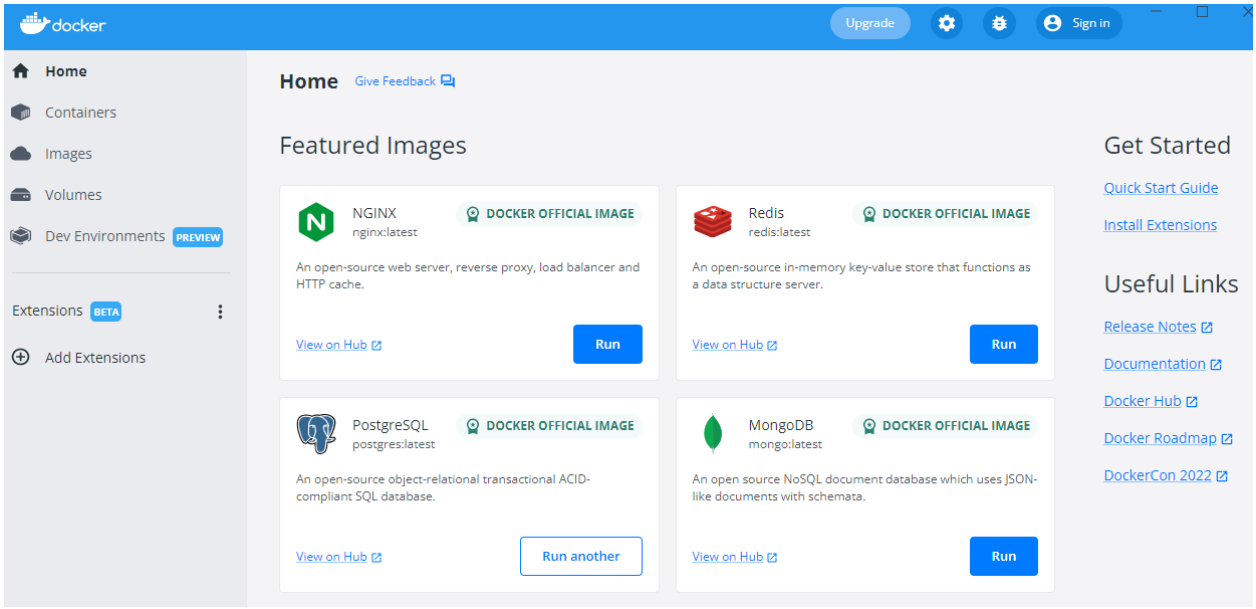
-просте налаштування середовища

-швидкий запуск контейнера

-створювати середовище можна на різних ОС

2.6 Docker Desctop

Docker Desctop – додаток який дозволяє створити контейнерні програми та мікрослужби. Головною перевагою є наявність шаблонів та образів у Docker Hub. Наприклад є можливість швидко розгорнути контейнер з Mysql , Mongodb, тощо. Інтерфейс додатку зображений на рис. 2.4



«Рисунок 2.4 – Інтерфейс застосунку Docker Desctop»

3 Етапи моделювання проекту

3.1 Опис етапів

Виходячи з поставленої задачі я визначив для себе 3 етапи:

1)моделювання схеми переходу кнопок

Основним об’єктом взаємодії з користувачем в моєму застосунку є кнопки, тому важливо було спроектувати кнопки таким чином, щоб користувачу було інтуїтивно зрозуміло що потрібно натиснути щоб виконати потрібну йому дію.

2)опис функціоналу кнопок

Далі потрібно визначитися з функціоналом кнопок, тобто що саме повинно відбутися після натискання.

3)Побудова діаграми відношень сутностей проекту

Останнім етапом було створення ERD бази даних. Необхідно було визначитися що саме потрібно зберігати, та які саме повинні бути зв’язки для того щоб кожен користувач міг отримати інформацію яка відповідає саме йому.

3.2 Побудування схеми переходу кнопок

На рис. 3.1 Графічно зображений перехід між кнопками. Потрібно відмітити, що при старті бота з’являється перше меню кнопок , яке включає в себе такі кнопки як: «Розклад», «Зв’язок зі старостою», «Список предметів» та «Список заходів» при натисканні меню кнопок змінюється відповідно до схеми яка намальована на рис. 3.1. Також, потрібно відмітити що з кожного меню можна повернутися до попереднього за допомогою кнопки «Назад».

«Рисунок 3.1 – Схема переходу кнопок»

3.3 Опис функціоналу кнопок

Опис функціоналу кнопок:

а)Розклад

Якщо користувач натисне розклад , то повинні з’явитися кнопки з назвами будніх днів. Потім при виборі дня, та натисканні на кнопку з’явиться розклад на цей день.

б)Зв’язок зі старостою

При натисканні цієї кнопки з’явиться посилання на старосту та кнопка для відправки файлу. Відправлений файл збережеться в папці адміністратора. Я вважаю що це може бути зручно коли старості потрібно зібрати якісь файли у всіх студентів.

в)Список предметів

При натисканні цієї кнопки з’явиться меню зі списком предметів

г)Назва предмету

Це кнопка яка приймає значення назви предмету зі списку предметів. Кількість цих кнопок дорівнює кількості предметів користувача.

Після натискання з’являється інформація коли в твоїй лабораторній групі проходять заняття, та посилання на них

д)Лабораторна робота

Це кнопка яка приймає значення назви лабораторної зі списку лабораторних.

При натисканні користувач отримує файл з завданням та інформацію по ній.

е)Список заходів

При натисканні на цю кнопку відправляється список заходів з Google Calendar на найближчі 18 годин

3.4 Побудова діаграми відношень сутностей

Діаграма відношень сутностей показує як зв’язані елементи всередені бази даних, а саме сутності та їх зв’язок. На рис. 3.2 зображена діаграма для мого проекту.

Створена база даних має такі сутності:

а) users

Містить у собі базову інформацію про користувачів, таку як: ім’я, прізвище, нікнейм у телеграмі. Пов’язана з таблицею roles як багато до одного. Пов’язана з таблицею groups як багато до одного. Пов’язана з таблицею subject groups через проміжну таблицю subject_groups_users.

б) roles

Створена для розмежування прав для користувачів. Є 2 варіанти значень: староста та студент. Пов’язана з таблицею users як один до багатьох.

в) groups

Створена для розмежування списку предметів для кожного користувача в залежності від групи. Пов’язана з таблицею subjects через проміжну таблицю groups_ subject.

г) subjects

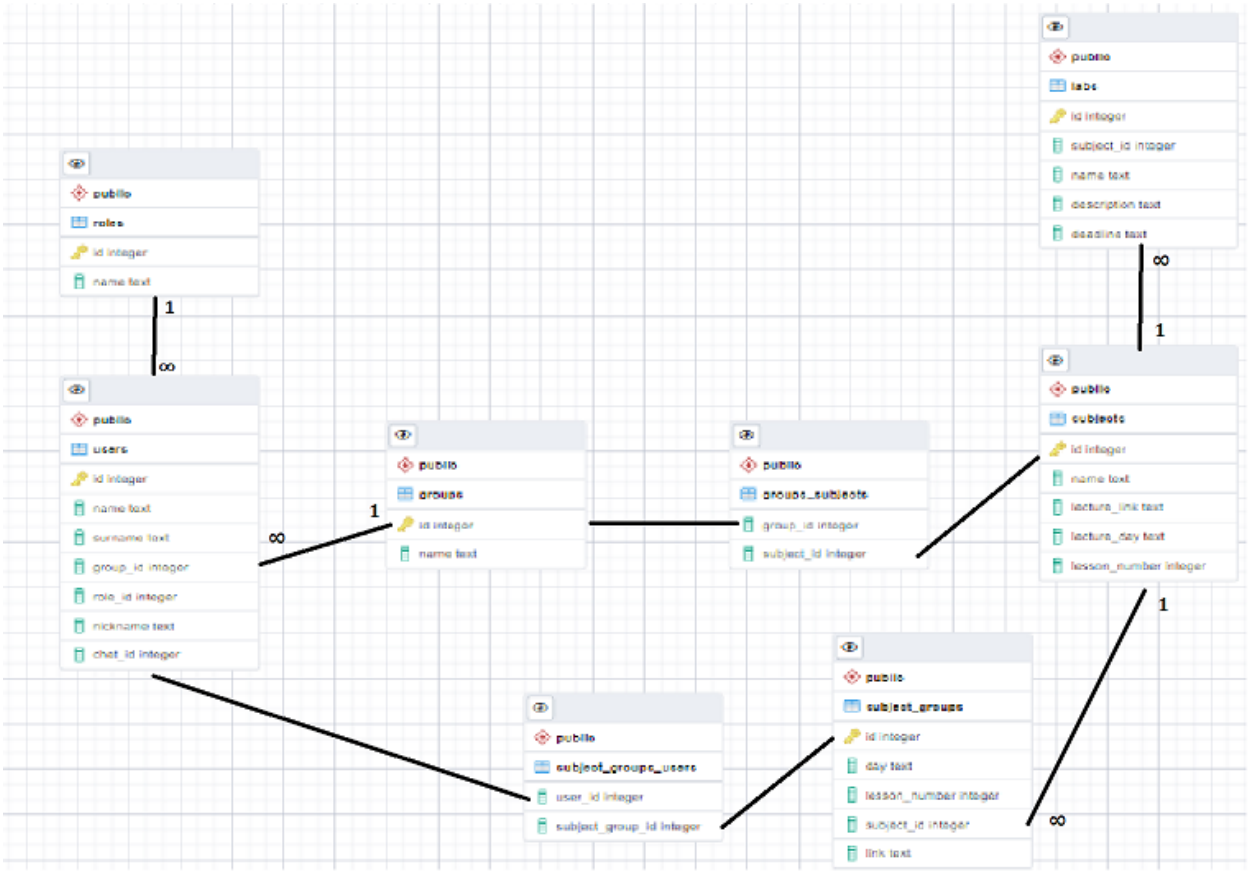
Містить у собі інформацію про предмет, таку як: назва, посилання на лекцію, день та номер пари на якій вона проводиться. Пов’язана з таблицею labs як один до багатьох. Пов’язана з таблицею subject_groups як один до багатьох.

д) labs

Містить інформацію про лабораторну роботу, таку як: назва, опис та терміни складання.

е)subject_groups

Містить інформацію про лабораторну групу, таку як: день проведення, номер пари та посилання.



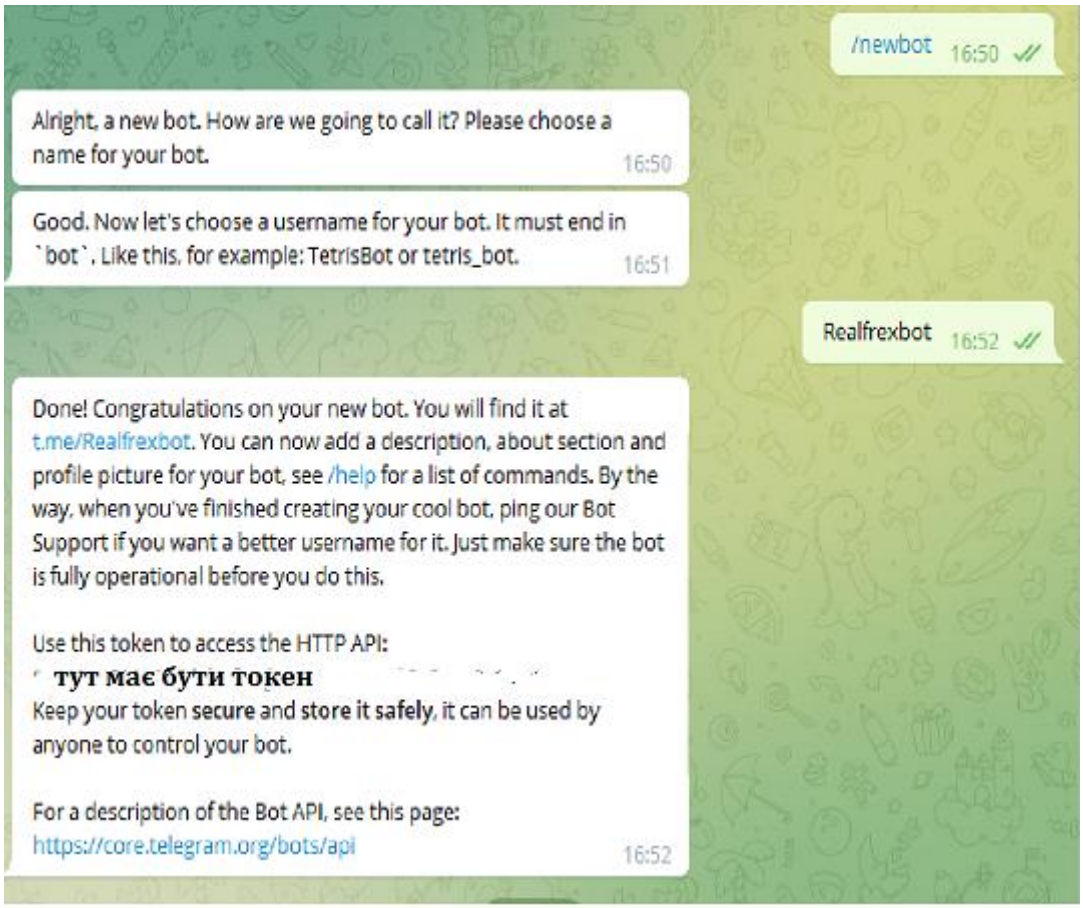
«Рисунок 3.2 – Діаграма відношень сутностей»

4 Розробка

4.1 Реєстрація бота

Спершу нам потрібно зареєструвати нашого бота. Для цього ми знаходимо телеграм бота BotFather. Один користувач може створити до 20 ботів.

«Рисунок 4.1 - BotFather»

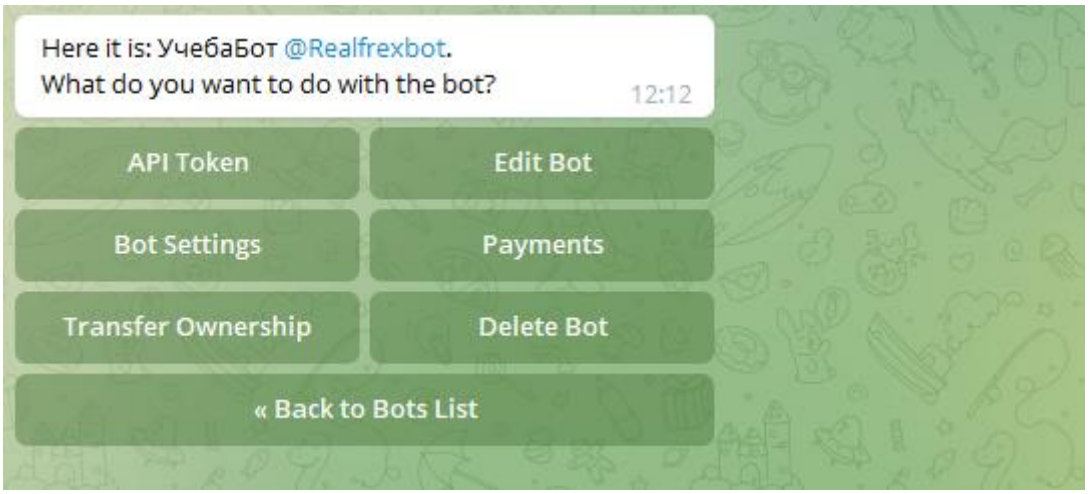


«Рисунок 4.2 - BotFather»

На рис. 4.2 можна побачити як я реєстрував свого бота. Спочатку я ввів команду «/newbot» для створення нового бота, потім вибрав назву та нікнейм бота. Після закінчення реєстрації BotFather надсилає нам унікальний токен, за допомогою якого сервери телеграм ідентифікують власника бота. На рисунку я замінив його рядком «тут має бути токен», для того щоб стороння людина не могла його використати.

Далі можна переглянути список ботів за допомогою команди «/mybots», як показано на рис. 4.3, та вибравши потрібного бота відкривається меню керування ботом, як показано на рис. 4.4. В цьому меню можемо отримати чи змінити токен, редагувати бота, додати платіжну систему, відкрити налаштування бота, змінити власника та видалити бота при необхідності.

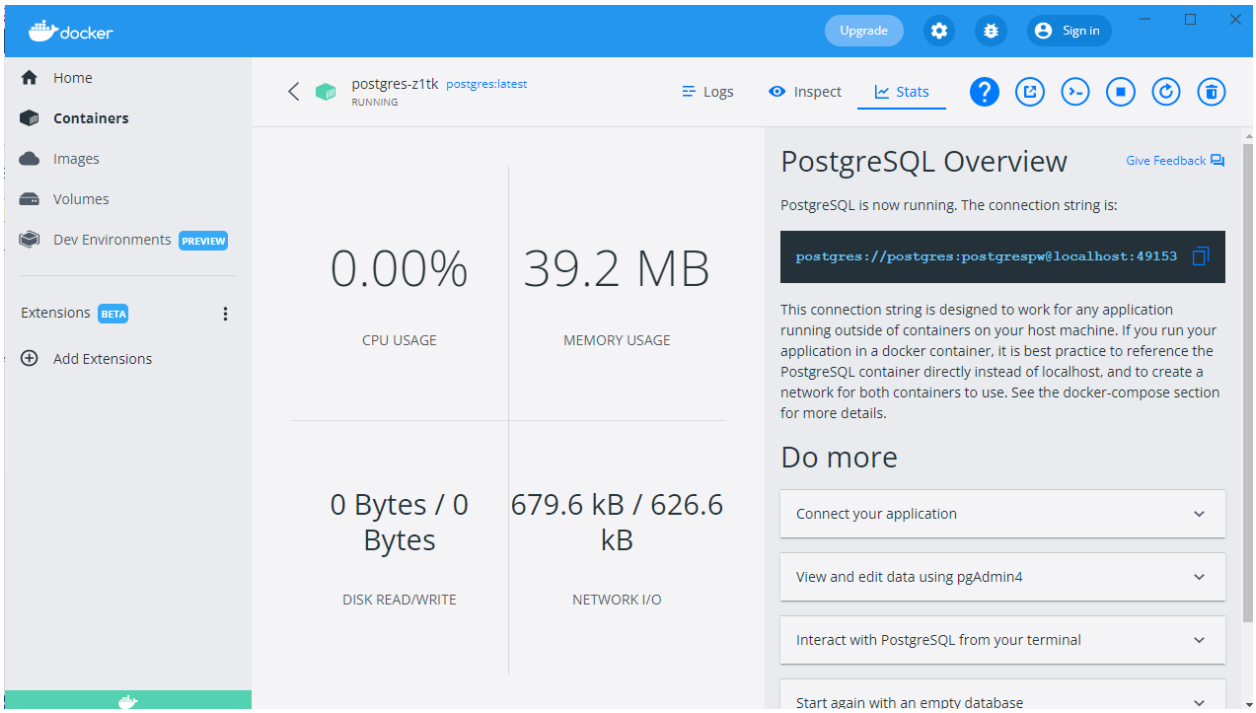
«Рисунок 4.3 – Команда «/mybost»»



«Рисунок 4.4 – Меню керування ботом»

4.2 Запуск PostgreSQL у контейнері

Для запуску сервера бази даних я вирішив використати Docker контейнери. На операційній системі Windows найпростіший спосіб це встановити програмне забезпечення «Docker Desctop» та використати вже готовий шаблон.



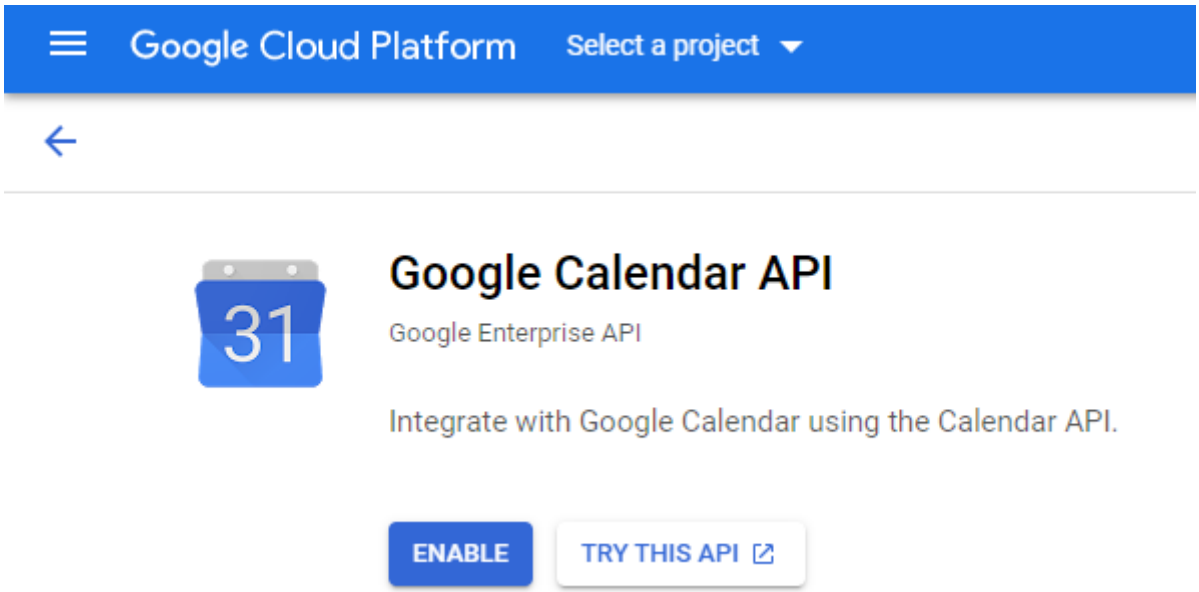
«Рисунок 4.5 – Запуск PostgreSQL у докер контейнері»

Цей додаток має простий інтерфейс та весь необхідний функціонал, такий як статистика, журнал та загальна інформація. На рис. 4.5 ми можемо бачити статистику. Відмічу те, що для запуску сервера мені знадобилося лише 39.2 MB пам’яті . Тому я вважаю що контейнеризація - це найкраще рішення для розробки мого проекту.

4.3 Взаємодія з Google Calendar API

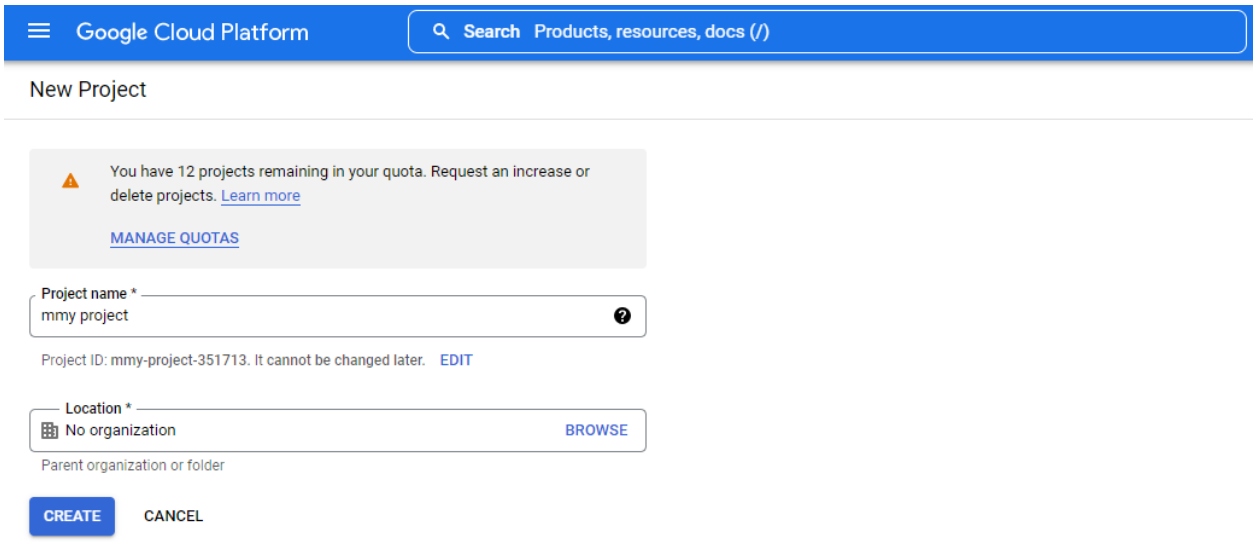
Під час проектування чат боту постала задача , як саме додавати та отримувати додані заходи. Зручним сервісом для планування є «Google Calendar» , який дозволяє створювати заходи на потрібний час та переглядати їх список на будь-якому пристрою на якому э доступ до облікового запису. Отже було вирішено синхронізувати роботу бота з подіями у календарі. Це можна реалізувати за допомогою Google Calendar API, який буде відповідати на наші HTTP-запити. Цей API надає більшість функцій веб версії календаря.

Для цього вмикаємо Google Calendar API на потрібному обліковому записі google як показано на рис. 4.6.



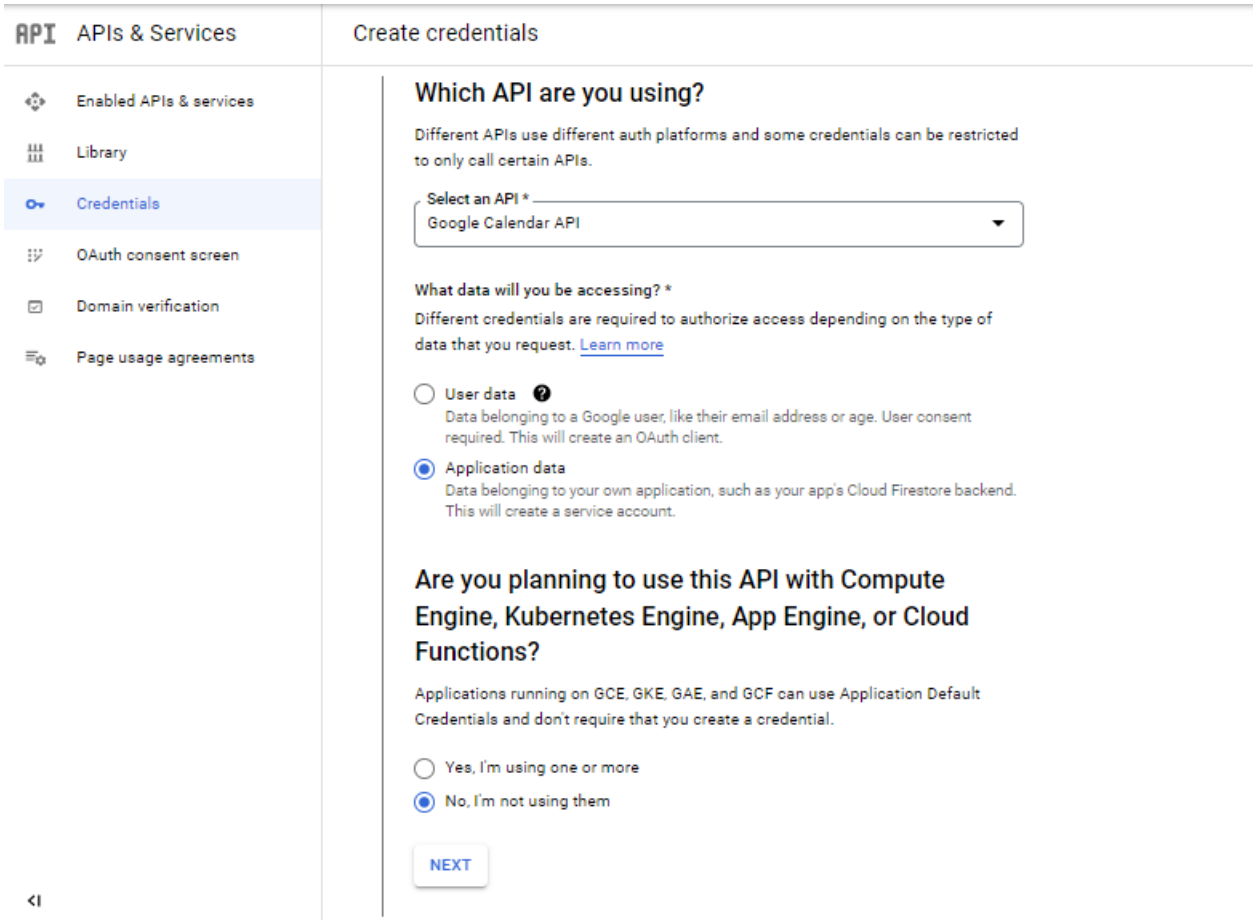
«Рисунок 4.6 – Приклад увімкненого Google Calendar API»

Далі створюємо новий проект як показано на рис. 4.7. Це необхідно для взаємодії з календарем, ми можемо створити декілька проектів для того щоб розділяти доступ до календаря для різних додатків.

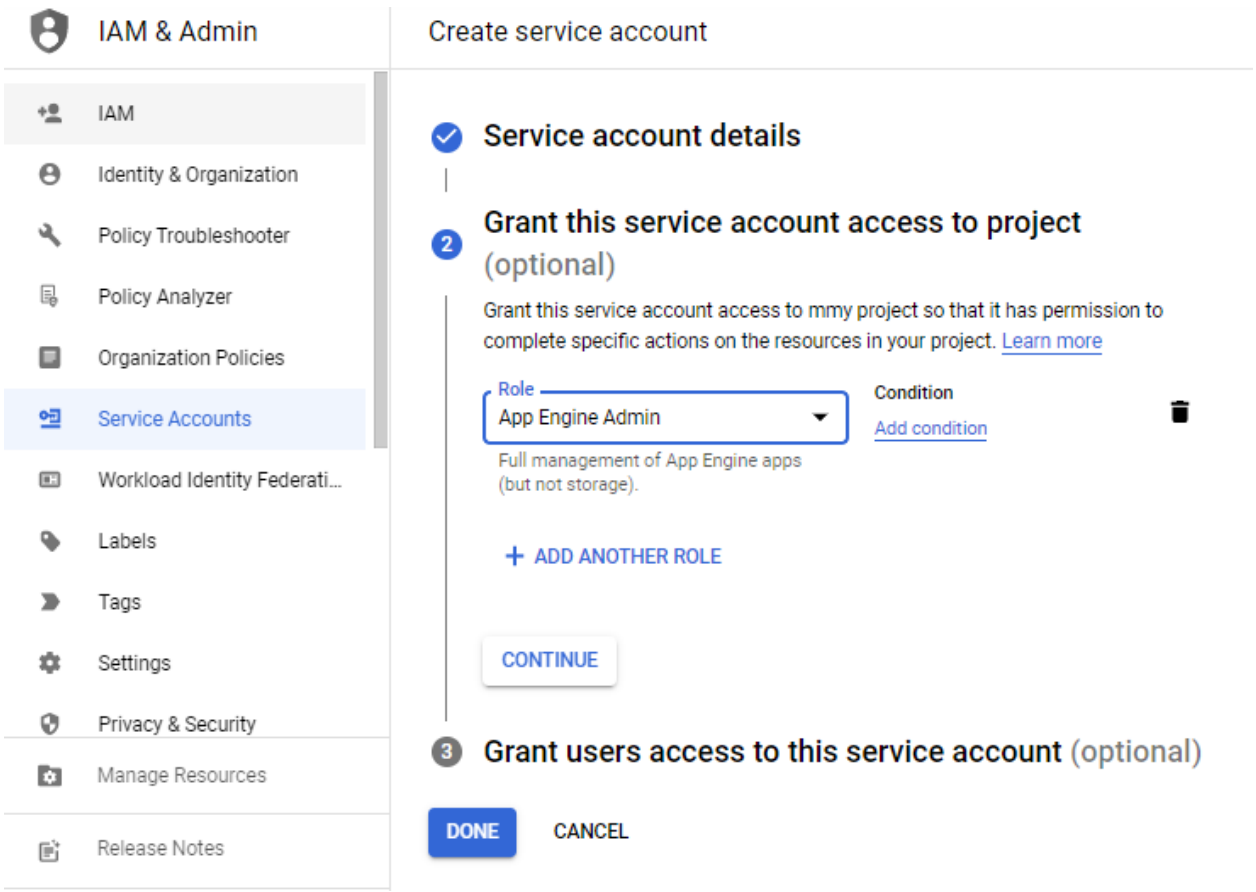


«Рисунок 4.7 – Створення проекту Google API»

Далі створюємо сервісний обліковий запис. Він потрібен для того, щоб до календаря міг під’єднатися бот. Це показано на рис. 4.8 , 4.9 .



«Рисунок 4.8 – Реєстрація сервісного облікового запису»



«Рисунок 4.9 – Вибір прав доступу»

Далі генеруємо закритий ключ, який використовується для дешифровки повідомлень. Це демонструється на рис. 4.10

«Рисунок 4.10 – Створення приватного ключа»

Для роботи з календарем за допомогою python потрібно встановити бібліотеку google-api-python-client. Після налаштувань пишемо код, за допомогою якого будуть надсилатися запити на отримання подій на найближчі 18 годин. Події ми отримуємо в вигляді словника Python, який містить у собі їх перелік та всю відому інформацію про них. Звісно, немає сенсу виводити всю інформацію тому, що звичайним користувачам немає необхідності знати дату створення події, дату зміни події, тощо. На мою думку найнеобхідніші дані для студента це: час події, назва, опис та посилання на подію. Для цього я написав окремий метод який наведено в додатку А під назвою «events».

4.4 Обробка команд за допомогою бібліотеки pyTelegramBotApi

Для запуску обробника команд за допомогою даної бібліотеки використовуються декоратори як наведено в прикладі на рис. 4.11. В данному випадку якщо ввести команду старт , то викличиться функція start.Далі вже можна прописувати логіку програми . В моєму випадку функція старт створює всі необхідні кнопки для початку роботи з ботом та разом з повідомленням «Привіт користувач» відображає їх.

```
@bot.message_handler(commands=['start'])
def start(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    i1 = types.KeyboardButton('Розклад')
    i2 = types.KeyboardButton("Зв'язок зі старостою")
    i3 = types.KeyboardButton('Список предметів')
    i4 = types.KeyboardButton('Список заходів')
    markup.add(i1, i2, i3, i4)
    bot.send_message(message.chat.id, 'Привіт, {0.first_name}!'.format(message.from_user), reply_markup=markup)
```

«Рисунок 4.11 – приклад обробки команд»

Окрім commands також є такі фільтри: content_types, regexp, chat_types, func. В своїй роботі я також використовував фільтр content_types, для написання логіки програми в залежності від натиснутої кнопки. На рис. 4.12 показаний приклад обробки якщо натиснути кнопку «Розклад». Повна логіка програми знаходиться в додатку А.

```
@bot.message_handler(content_types=['text'])
def bot_message(message):
    if message.chat.type == 'private':
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        if message.text == 'Розклад':
            item1 = types.KeyboardButton('Понеділок')
            item2 = types.KeyboardButton('Вівторок')
            item3 = types.KeyboardButton('Середа')
            item4 = types.KeyboardButton('Четвер')
            item5 = types.KeyboardButton("П'ятниця")
            back = types.KeyboardButton('Назад в головне меню ')
            markup.add(item1, item2, item3, item4, item5, back)
            bot.send_message(message.chat.id, 'Тут знаходиться актуальний розклад ', reply_markup=markup)
```

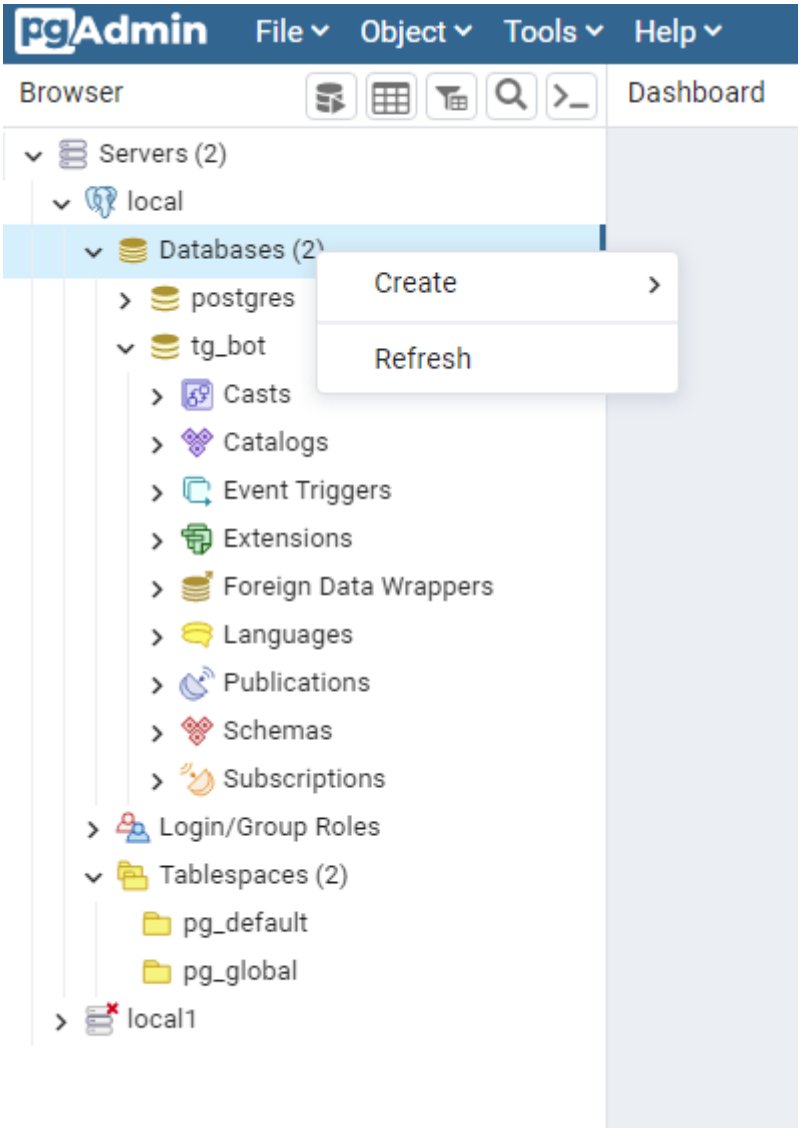
«Рисунок 4.12 - Приклад обробки кнопки «Розклад»»

4.5 Створення бази даних в застосунку pgAdmin

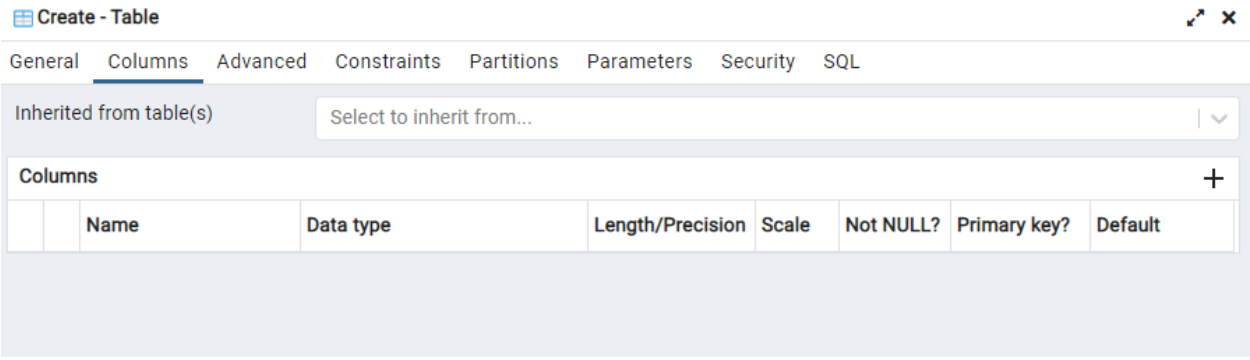
pgAdmin – платформа для зручної розробки та адміністрування

СУБД PostgreSQL , я обрав її для швидкої та ефективної розробки бази даних та подальшого наповнення її даними.

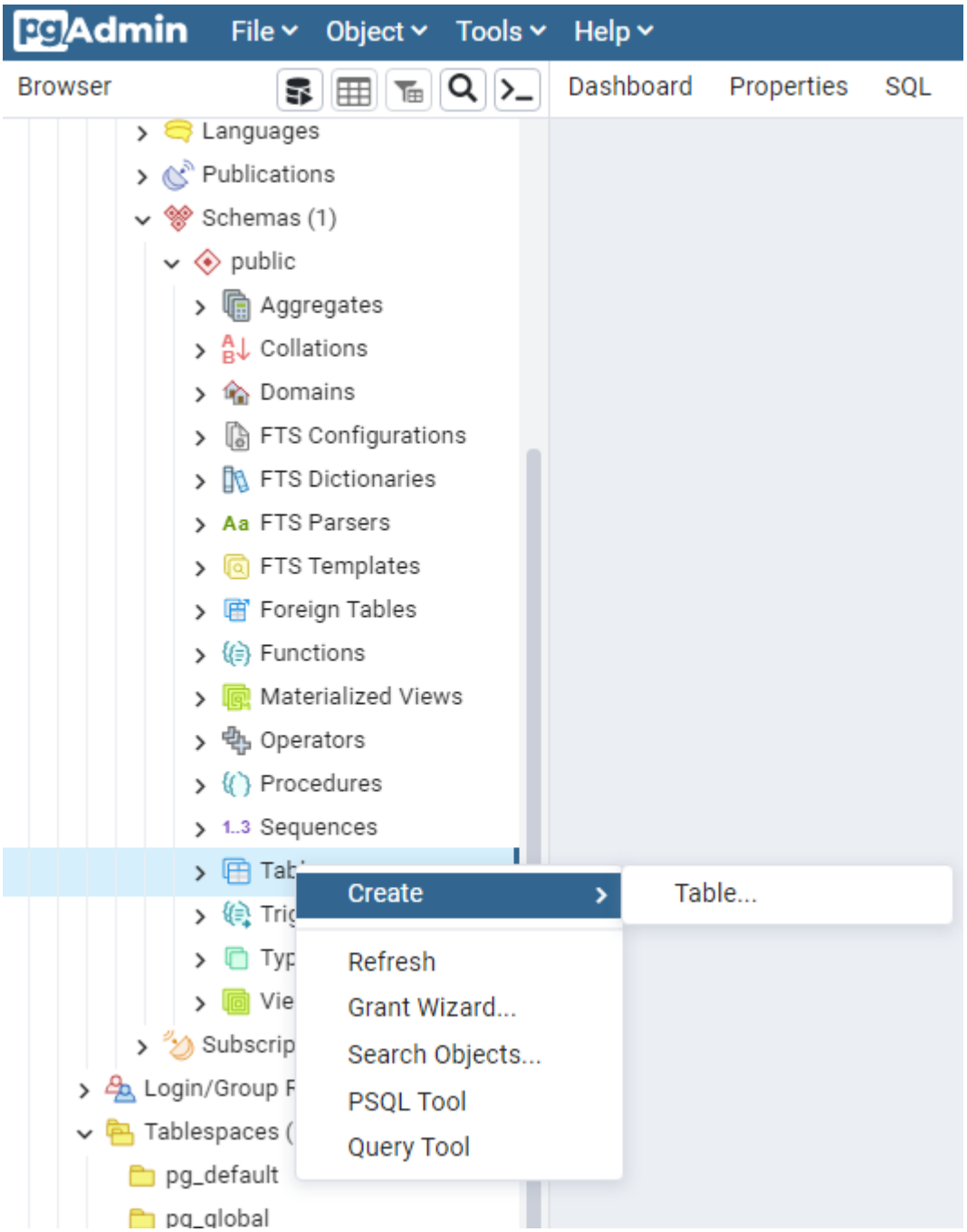
Спочатку створюємо базу даних як показано на рис. 4.13 . Наступним етапом буде створення таблиць та полів як показано на рис 4.15 та 4.14 відповідно. На рис. 4.16 показані всі таблиці які були створені для реалізації мого проекту.



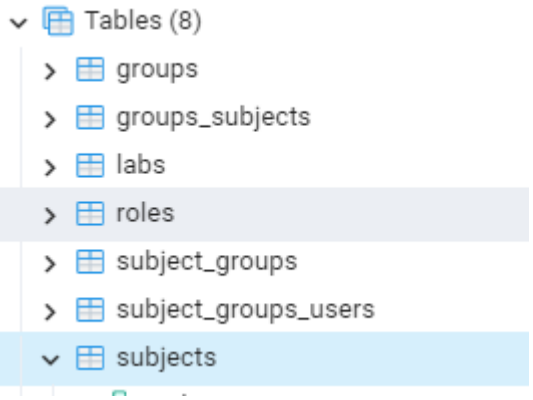
«Рисунок 4.13 – Створення бази даних»



«Рисунок 4.14 – Створення полів»



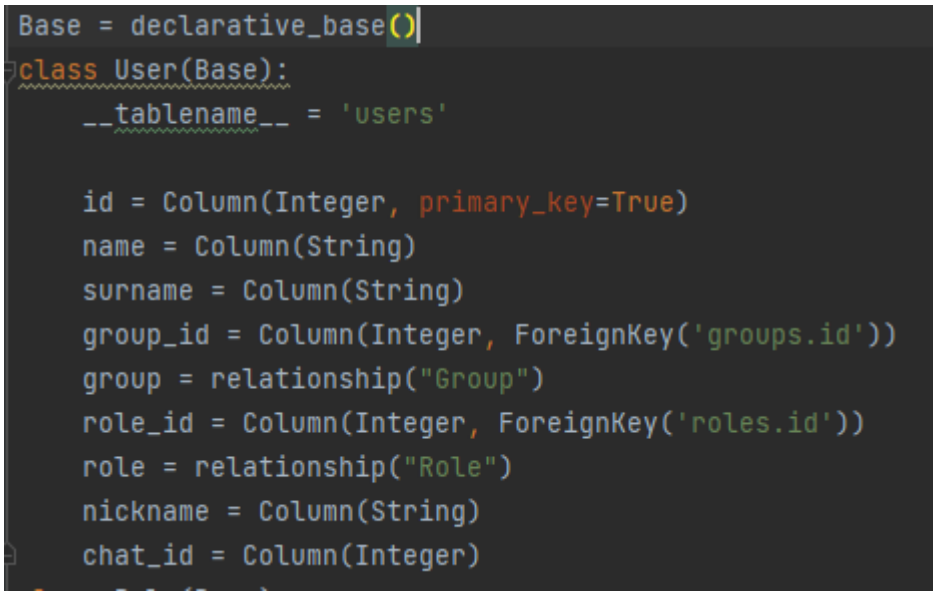
«Рисунок 4.15 – створення таблиці»



«Рисунок 4.16 – Список створених таблиць»

4.6 Представлення таблиць бази даних за допомогою Sql Alchemy

Для початку роботи потрібно представити таблиці бази даних у вигляді класів Python. Клас має бути успадкований від базового класу за допомогою виклику declarative_base функції. Також має обов'язково бути об'явлене ім'я таблиці та первинний ключ. На рис. 4.16 наведений приклад реалізації класу User який відповідає таблиці users у розробленій базі даних.



«Рисунок 4.16 – Клас User»

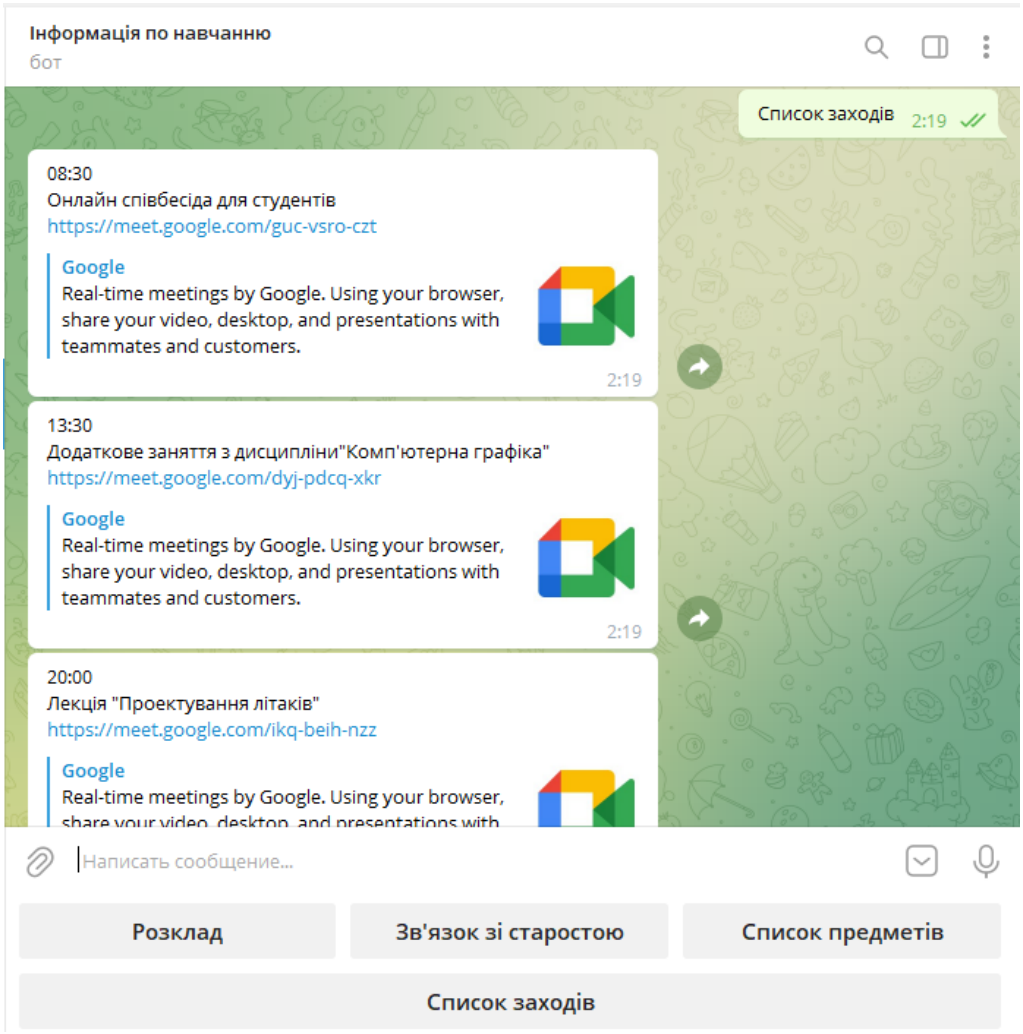
Поля бази даних задаються за допомогою функції Column(). Зв'язки задаються за допомогою функції relationship(). Далі для того щоб почати працювати з базою даних потрібно створити сесію за допомогою функції Session(). Подальша взаємодія проходить з об'єктами створеного класу. Всі створені класи знаходяться в додатку Б. Приклади звернення до бази даних знаходяться в додатку А.

5 Огляд та тестування

5.1 Огляд функціоналу створеного бота

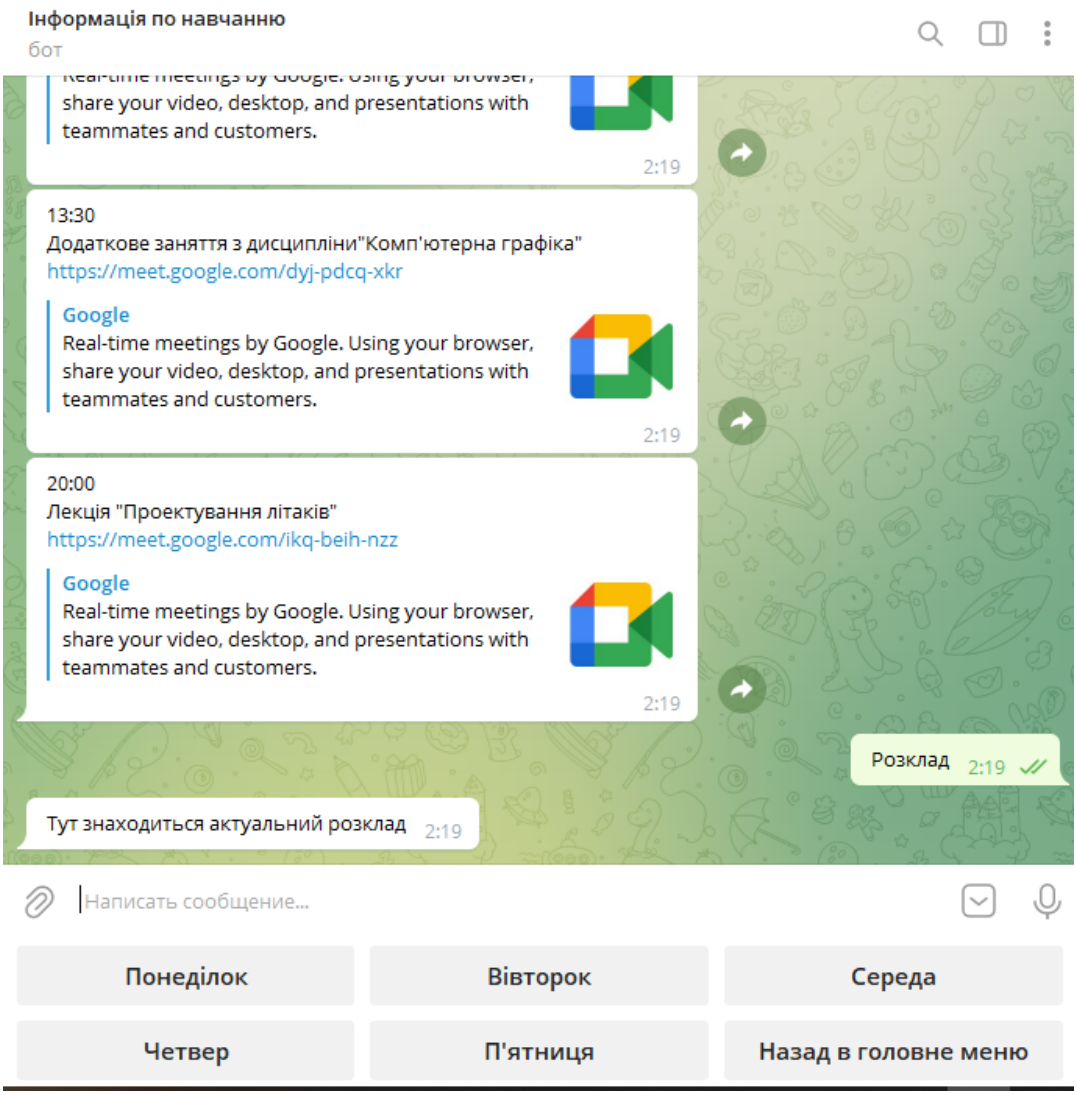
Нижче на рисунках представлений інтерфейс а також функціонал телеграм бота. Для зручності користувачів уся взаємодія відбувається за допомогою інтерактивних кнопок.

На рис. 5.1 показаний вивід заходів з Google Calendar який відправляється у чат після натискання кнопки «Список заходів». Виводиться інформація на день коли робиться цей запит. Перший рядок – це час проведення, другий – назва, третій – посилання.

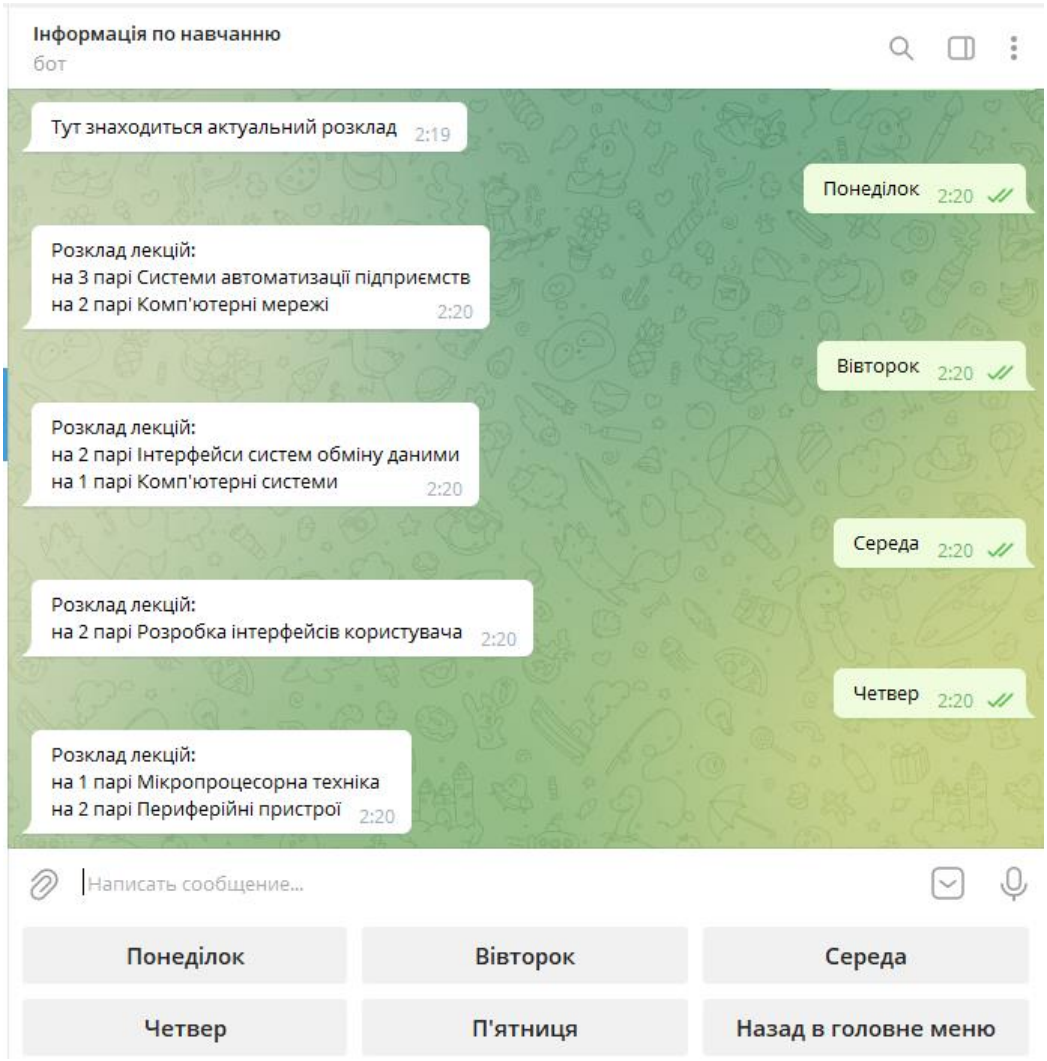


«Рисунок 5.1 – вивід списку заходів»

На рис. 5.2 показано що відбувається після натискання кнопки «Розклад». Після натискання на цю кнопку з’являються кнопки для вибору дня тижня на який потрібний розклад. На рис. 5.3 показано що відбувається при натисканні на кнопку з назвою дня тижня. Після натискання на цю кнопку користувачеві приходить повідомлення з розкладом, в залежності від групи в якій він знаходиться.

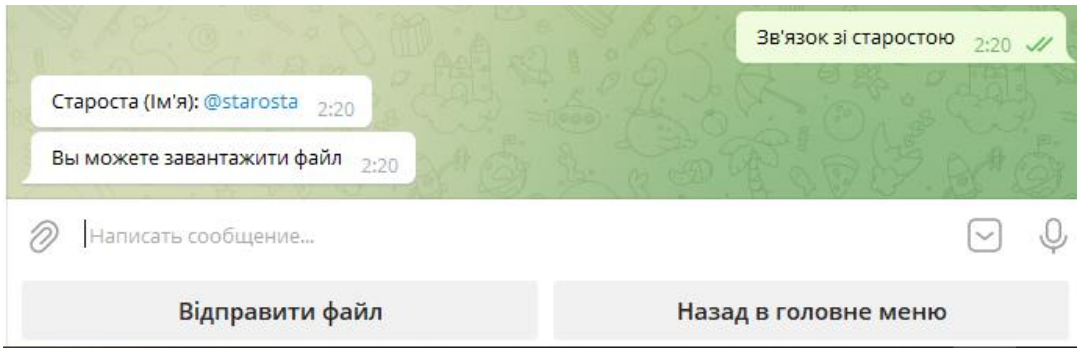


«Рисунок 5.2 – Розклад по дням тижня»

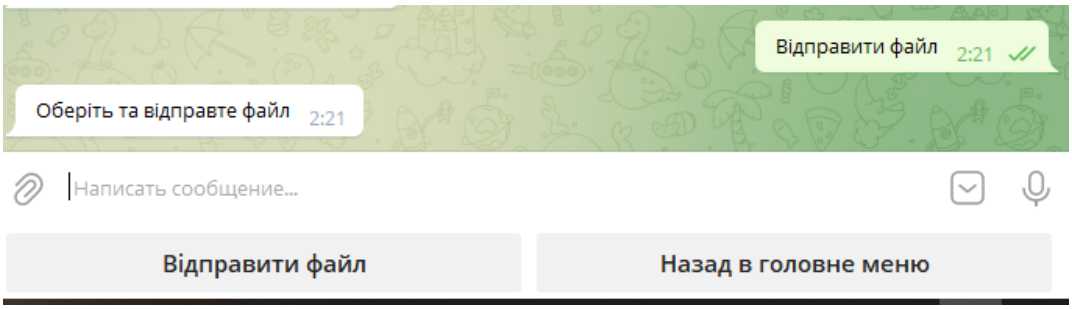


«Рисунок 5.3 – приклад виводу розкладу»

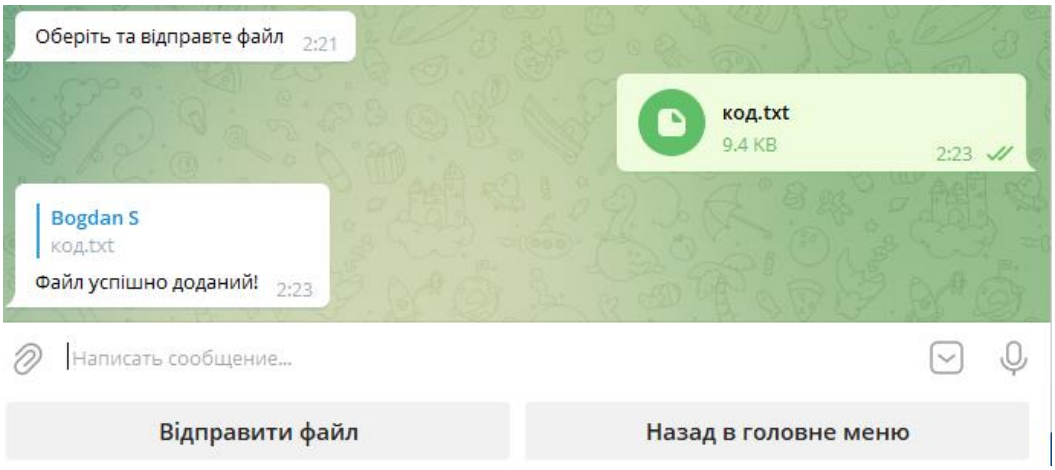
На рис. 5.4 показано що відбувається при натисканні на кнопку «зв’язок зі старостою». Після натискання на цю кнопку з’являється посилання на старосту, та можливість відправити файл. На рис 5.5 та 5.6 показано спосіб відправки файлу на локальний комп’ютер адміністратора.



«Рисунок 5.4 – Зв’язок зі старостою»

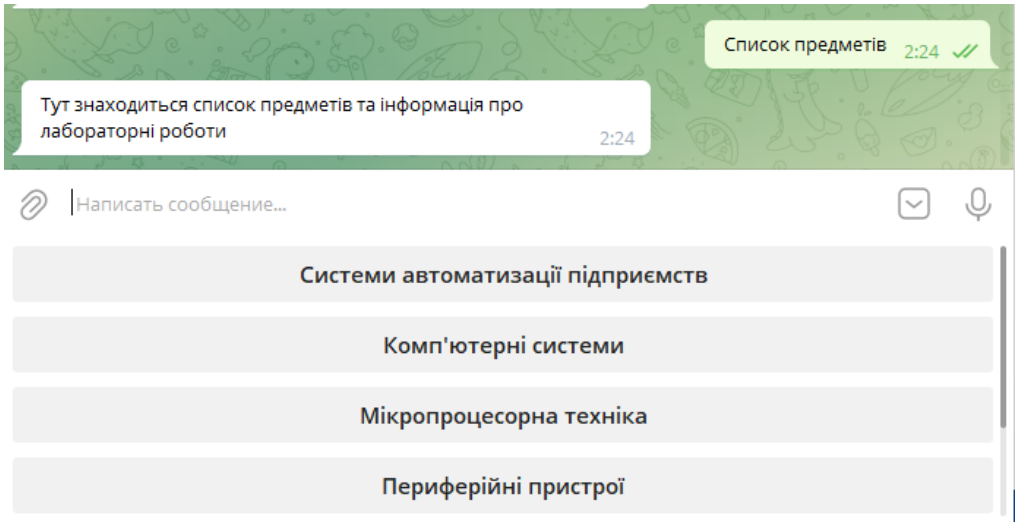


«Рисунок 5.5 – Відправка файлу»

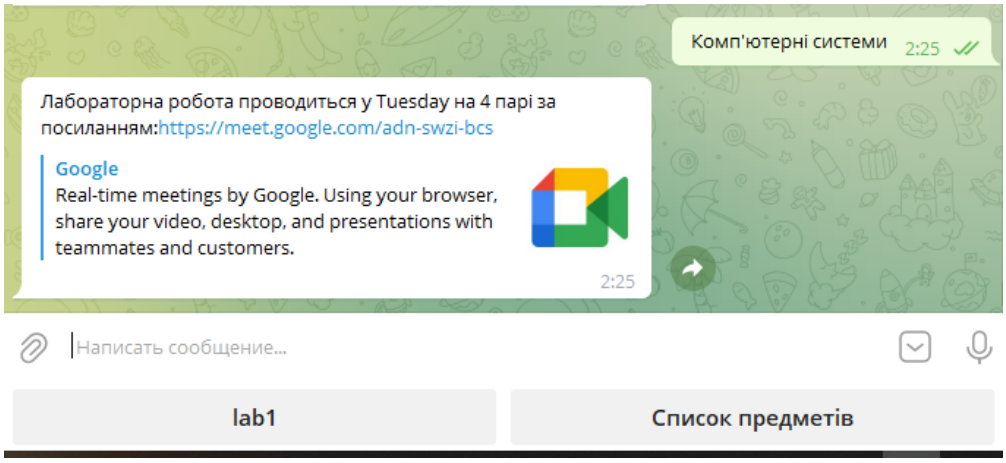


«Рисунок 5.6 – успішно відправлений файл»

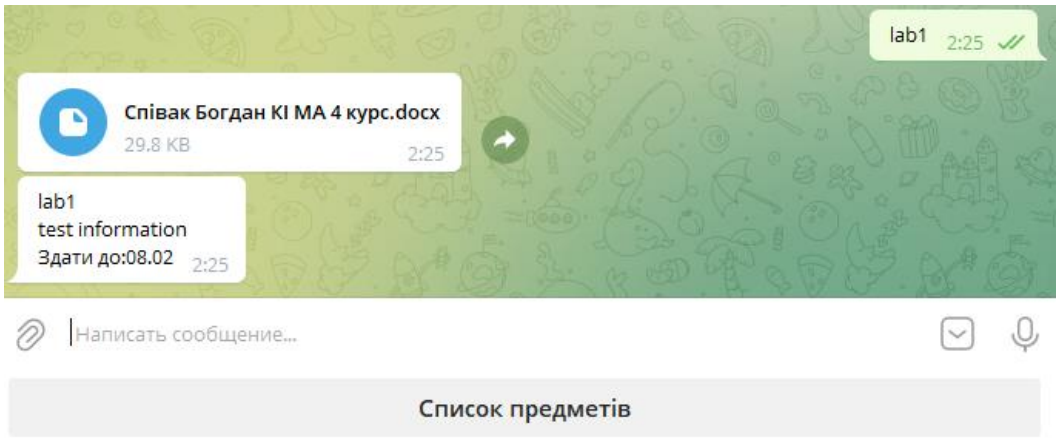
На рис. 5.7 показано вивід списку предметів користувача. Тут виводяться предмети які відповідають групі студента. На рис. 5.8 показано що при натисканні на назву предмету виводиться інформація про лабораторну групу студента та посилання на прийом лабораторних. На рис. 5.9 показано що при натисканні на кнопку з номером лабораторної в чат відправляється завдання та інформація про лабораторну роботу.



«Рисунок 5.7 – Список предметів»



«Рисунок 5.8 – Список лабораторних робіт»



«Рисунок 5.9 – Інформація про вибрану лабораторну роботу»

5.2 Тестування

Першим етапом мого тестування було наповнення бд даними та перевірка працездатності боту з різних облікових записів. Метою цього тесту було перевірити правильність взаємодії бази даних та самої програми. Цей тест пройшов успішно , всі данні які я отримував були вірні.

Метою наступного етапу було зрозуміти чи правильно я продумав інтерфейс та функціонал бота, тому запросив групу студентів спробувати скористатися ботом. Протягом тижня замість звичних для них методів

отримання інформації використовували телеграм бота. Цей тест також був пройдений успішно, функціонал був оцінений як корисний, а взаємодія з ботом була легкою та інтуїтивно зрозумілою.

Наступним та останнім етапом було тестування продуктивності. Групі студентів було запропоновано одночасно посилати запити боту для тестування часу відгуку. Основною метою було зрозуміти чи буде затримка прийнятною для звичайних користувачів. Зважаючи на те що цей бот при написанні розраховувався на 70 – 100 людей великої кількості одночасних запитів не повинно виникати, але все ж під час тестування створилися саме такі рідкісні умови. Під час одночасного користування, затримки досягали 3х секунд, але втрати повідомлень не спостерігалося. Узагальнюючи думку студентів , які брали участь , можу зробити висновок , що швидкість відповіді є прийнятною.

Отже, після тестування можна зробити висновки, що весь функціонал працює належним чином та є актуальним для цільової аудиторії. Інтерфейс є легким для сприйняття та є інтуїтивно зрозумілим. Швидкість роботи є прийнятною для тієї кількості людей на яку розраховувався телеграм бот.

Висновки

Отже, було розроблено механізм простого доступу до всієї необхідної інформації для студента. Було досліджено можливості чат ботів, їх особливості, переваги та недоліки.

Було реалізовано доступ до подій з google calendar, що надає користувачеві можливість переглядати плани на день не витрачаючи на це додатковий час. Також, було реалізовано зберігання в базі даних всієї необхідної інформації про навчальний процес, для подальшого її використання. Ще однією важливою частиною була розробка зручного інтерфейсу , для того щоб користувач міг інтуїтивно знайти те що йому потрібно.

Перевагами мого бота можна вважати:

-доступність у будь-який час

-швидке отримання відповіді

-легкість в масштабуванні, для того щоб бот працював у різних групах, достатньо згенерувати новий токен, та просто замінити його у коді програми

-дуже простий та зрозумілий інтерфейс користувача

-не потрібно нічого додатково встановлювати

Можна зробити висновок, що чат боти - це простий та ефективний інструмент для розробки будь якого функціоналу, починаючи від систем моніторингу, закінчуючи повноцінними магазинами. Завдяки тому, що не потрібно витрачати час на розробку інтерфейсу користувача можна повністю зосередитись на поставленій задачі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Статистика месенджерів [Електронний ресурс]

https://www.pravda.com.ua/rus/news/2021/11/19/7314556/

2. Telegram Bot API [Електронний ресурс]

https://stackshare.io/telegram-bot-api

3. Telegram Bot API [Електронний ресурс]

https://core.telegram.org/bots/api

4. Боти[Електронний ресурс]

https://ru.wikipedia.org/wiki/Бот_(программа)#Чатботы

5. Чат-боти[Електронний ресурс]

https://startupbonsai.com/chatbot-statistics/

6. Довідник Bot API [Електронний ресурс]

https://tlgrm.ru/docs/bots/api#authorizing-your-bot

7. PostgreSQL [Електронний ресурс]

https://www.postgresql.org/docs/

8. Документація по телеграм ботам [Електронний ресурс]

https://tlgrm.ru/docs/bots#supersposobnosti

9. Довідник Python

https://docs.python.org/3/tutorial/index.html [Електронний ресурс]

10. pyTelegramBotAPI [Електронний ресурс]

https://github.com/eternnoir/pyTelegramBotAPI

11. SqlAlchemy [Електронний ресурс]

https://docs.sqlalchemy.org/en/14/orm/basic_relationships.html

11. Drift State of Conversational Marketing [Електронний ресурс]

https://www.drift.com/blog/state-of-conversational-marketing/?utm_source=salesforce&utm_medium=blog

12. Детектор медіа[Електронний ресурс]

https://ms.detector.media/sotsmerezhi/post/28531/2021-11-19-opytuvannya-viber-facebook-messenger-i-telegram-naypopulyarnishi-mesendzhery-sered-ukraintsiv/

13.Docker [Електронний ресурс]

https://aws.amazon.com/ru/docker/

14. Telegram API [Електронний ресурс]

https://www.servicedesk.site/en/2019/08/09/як-створити-телеграм-бота-правильно/

15. MTproto [Електронний ресурс]

https://docs.pyrogram.org/topics/mtproto-vs-botapi

Додаток А

Код застосунку написаний мовою python:

```
import telebot
from models import User,Role,Group,Groups_subject,Subject,Lab,Subject_Group
from telebot import types
import httplib2
import config
from sqlalchemy.orm import declarative_base,sessionmaker
from sqlalchemy import create_engine
import datetime
import time
from apiclient import discovery
from oauth2client.service_account import ServiceAccountCredentials
TOKEN = config.TOKEN  # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
bot = telebot.TeleBot(TOKEN)
engine =
create_engine('postgresql+psycopg2://postgres:postgrespw@localhost:49153/tg
_bot')
Session = sessionmaker(bind=engine)
session = Session()
Base = declarative_base()
```

```
@bot.message_handler(commands=['start'])
def start(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    i1 = types.KeyboardButton('Розклад')
    i2 = types.KeyboardButton("Зв'язок зі старостою")
    i3 = types.KeyboardButton('Список предметів')
    i4 = types.KeyboardButton('Список заходів')
    markup.add(i1, i2, i3, i4)
    bot.send_message(message.chat.id, 'Привіт,
{0.first_name}!'.format(message.from_user), reply_markup=markup)

@bot.message_handler(content_types=['text'])
def bot_message(message):
    if message.chat.type == 'private':
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        l1 = return_subjects(message.chat.id)
        if message.text == 'Розклад':
            i1 = types.KeyboardButton('Понеділок')
            i2 = types.KeyboardButton('Вівторок')
            i3 = types.KeyboardButton('Середа')
            i4 = types.KeyboardButton('Четвер')
            i5 = types.KeyboardButton("П'ятниця")
```

```
        back = types.KeyboardButton('Назад в головне меню')
        markup.add(i1, i2, i3, i4, i5, back)
        bot.send_message(message.chat.id, 'Тут знаходиться актуальний розклад ', reply_markup=markup)
    elif message.text == "Зв'язок зі старостою":
        bot.send_message(message.chat.id, "Староста (Ім'я): @starosta")
        i1 = types.KeyboardButton('Відправити файл')
        back = types.KeyboardButton('Назад в головне меню')
        markup.add(i1, back)
        bot.send_message(message.chat.id, 'Вы можете завантажити файл ', reply_markup=markup)
    elif message.text == "Список заходів":
        events()
    elif message.text == "Список предметів":
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        back = types.KeyboardButton('Назад в головне меню')
        for x in range(len(l1)):
            item = types.KeyboardButton(l1[x])
            if x == len(l1) - 1: markup.add(item,back)
            else:
                markup.add(item)
```

```
        bot.send_message(message.chat.id, 'Тут знаходиться список предметів та інформація про лабораторні роботи ', reply_markup=markup)
    elif message.text == 'Назад в головне меню□':
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        i1 = types.KeyboardButton('Розклад')
        i2 = types.KeyboardButton("Зв'язок зі старостою")
        i3 = types.KeyboardButton('Список предметів')
        i4 = types.KeyboardButton('Список заходів')
        markup.add(i1, i2, i3, i4, )
        bot.send_message(message.chat.id, 'Ви можете знайти тут відповіді на ваші запитання по навчанню',
                         reply_markup=markup)
    elif message.text == 'Понеділок':
        bot.send_message(message.chat.id, return_schedule('Monday'))
    elif message.text == 'Вівторок':
        bot.send_message(message.chat.id, return_schedule('Tuesday'))
    elif message.text == 'Середа':
        bot.send_message(message.chat.id, return_schedule('Wednesday'))
    elif message.text == 'Четвер':
        bot.send_message(message.chat.id, return_schedule('Thursday'))
    elif message.text == "П'ятниця":
        bot.send_message(message.chat.id, return_schedule('Friday'))
```

```
    elif message.text == 'Відправити файл':
        msg = bot.send_message(message.chat.id, "Оберіть та відправте файл")
        bot.register_next_step_handler(msg, save_doc)
    labs = session.query(Lab).all()
    for x in range(len(l1)):
        if message.text == l1[x]:
            information = return_lab_group_info(message.chat.id,
get_subject_id(l1[x]))
            for lab in labs:
                if lab.subject_id == get_subject_id(l1[x]):
                    item = types.KeyboardButton(lab.name)
                    back = types.KeyboardButton("Список предметів")
                    markup.add(item, back)
                    bot.send_message(message.chat.id, information,
reply_markup=markup)
    for lab in labs:
        text = lab.name + "\n" + lab.description + "\n" "Здати до:" + lab.deadline
        if message.text == lab.name:
            back = types.KeyboardButton("Список предметів")
            markup.add(back)
            f = open("D:\\Співак Богдан КІ МА 4 курс.docx", "rb")
            bot.send_document(message.chat.id, f)
```

```
            bot.send_message(message.chat.id, text, reply_markup=markup)
def save_doc(message):
    if message.chat.type == 'private':
        try:
            file_inf = bot.get_file(message.document.file_id)
            file_upload = bot.download_file(file_inf.file_path)
            path = 'D:/' + message.document.file_name
            with open(path, 'wb') as new_file:
                new_file.write(file_upload)
            bot.reply_to(message, "Файл успішно доданий!")
        except Exception as e:
            bot.reply_to(message, 'Вы не обрали файл!')
def after_text_2(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    text =[]
    texts = [message.text]
    for text in texts:
        print(text)
        print("vot tak")
def return_subjects(message_chat_id):
    users = session.query(User).all()
```

```
    subjects = []
    for user in users:
        if user.chat_id == message_chat_id:
            group_subjects = user.group.subjects
            for group_subject in group_subjects:
                subject =group_subject.subject.name
                subjects.append(subject)
    return subjects
def get_subject_id(name):
    subjects = session.query(Subject).all()
    for subject in subjects:
        if subject.name == name:
            return subject.id
def return_schedule(day):
    subjects = session.query(Subject).all()
    schedule = "Розклад лекцій: \n"
    for subject in subjects:
        if subject.lecture_day == day:
            schedule +="на " + str(subject.lesson_number) + " парі "+
str(subject.name) +"\n"
    if schedule != "Розклад лекцій: \n": return schedule
    else: return "Лекції відсутні"
```

```
def return_lab_group_info(chat_id,subject_id):
    users = session.query(User).all()
    for user in users:
        if user.chat_id == chat_id:
            user_subject_groups = user.subject_groups
            for user_subject_group in user_subject_groups:
                if (user_subject_group.subject_id == subject_id):
                    text = "Лабораторна робота проводиться у "
+user_subject_group.day + " на " + str(user_subject_group.lesson_number) + "
парі за посиланням:" + user_subject_group.link
                    print(text)
    return text
def events():
    credentials =
ServiceAccountCredentials.from_json_keyfile_name(config.client_secret_calenda
r,
'https://www.googleapis.com/auth/calendar.readonly')
    http = credentials.authorize(httplib2.Http())
    service = discovery.build('calendar', 'v3', http=http)
    now = datetime.datetime.utcnow().isoformat() + 'Z'  # 'Z'  UTC time
    hours18 = round(time.time()) + 57600
    hours18 = datetime.datetime.fromtimestamp(hours18).isoformat() + 'Z'
    print('джоб страр')
```

```
    eventsResult = service.events().list(
        calendarId='bodyaspiv@gmail.com', timeMin=now, timeMax=hours18,
maxResults=100,
        singleEvents=True,
        orderBy='startTime').execute()
    events = eventsResult.get('items', [])
    print(len(events))
    for x in range(len(events)):
        print(events[x])
    else:

        for event in events:
            start = event['start'].get('dateTime', event['start'].get('date'))
            start = start[11:-9]
            link = event.get('hangoutLink', "")
            print(start, '\n', event['summary'], link)
            description = event.get('description', "немає опису")
            print(description)
            bot.send_message(448319116, start + '\n' + event['summary'] + '\n' + link)

bot.polling(none_stop=True)
```

Додаток Б

Створення таблиць та відношень за допомогою класів:

```
from sqlalchemy import  Table, Column, Integer, String, ForeignKey,
create_engine

from sqlalchemy.orm import declarative_base, relationship,sessionmaker

engine =
create_engine('postgresql+psycopg2://postgres:postgrespw@localhost:49153/tg
_bot')

Session = sessionmaker(bind=engine)

session = Session()

Base = declarative_base()

subject_groups_users = Table('subject_groups_users', Base.metadata,

  Column('user_id', Integer(), ForeignKey("users.id")),

  Column('subject_group_id', Integer(), ForeignKey("subject_groups.id"))

)


class User(Base):

   __tablename__ = 'users'


   id = Column(Integer, primary_key=True)
```

```
    name = Column(String)
    surname = Column(String)
    group_id = Column(Integer, ForeignKey('groups.id'))
    group = relationship("Group")
    role_id = Column(Integer, ForeignKey('roles.id'))
    role = relationship("Role")
    nickname = Column(String)
    chat_id = Column(Integer)
class Role(Base):
    __tablename__ = 'roles'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    user = relationship("User")
print("proverka")
class Group(Base):
    __tablename__ = 'groups'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    subjects = relationship("Groups_subject",back_populates="group")
```

```
class Groups_subject(Base):
    __tablename__ = 'groups_subjects'

    group_id = Column(ForeignKey("groups.id"),primary_key=True)
    subject_id = Column(ForeignKey("subjects.id"),primary_key=True)
    group = relationship("Group",back_populates='subjects')
    subject = relationship("Subject",back_populates='groups')
class Subject(Base):
    __tablename__ = 'subjects'

    id = Column(Integer, primary_key=True)
    lecture_link = Column(String)
    name = Column(String)
    lecture_day = Column(String)
    lesson_number = Column(Integer)
    lab = relationship("Lab")
    subject_group = relationship("Subject_Group")
    groups = relationship("Groups_subject", back_populates="subject")
class Lab(Base):
    __tablename__ = 'labs'
```

```
    id = Column(Integer, primary_key=True)
    subject_id = Column(Integer, ForeignKey('subjects.id'))
    subject = relationship("Subject")
    name = Column(String)
    description = Column(String)
    deadline = Column(String)
class Subject_Group(Base):
    __tablename__ = 'subject_groups'


    id = Column(Integer, primary_key=True)
    day = Column(String)
    lesson_number = Column(Integer)
    subject_id = Column(Integer, ForeignKey('subjects.id'))
    subject = relationship("Subject")
    link = Column(String)
    # user_id = Column(Integer, ForeignKey('users.id'))
    user = relationship("User", secondary=subject_groups_users,
backref="subject_groups")
Session = sessionmaker(bind=engine)
session = Session()
users = session.query(User).all()
for user in users:
```

```
    if user.chat_id==782151498:
# pk=user.group_id
# group = session.query(Group).get(pk)
        user_subject_groups = user.subject_groups
        for user_subject_group in user_subject_groups:
          if( user_subject_group.subject_id == 2):
              print(user_subject_group.day)
              print(user_subject_group.lesson_number)
              print(user_subject_group.link)
        group_subjects= user.group.subjects
        for group_subject in group_subjects:
           print(group_subject.subject.name)
           print(group_subject.subject.id)
        for group_subject in group_subjects:
           labs=group_subject.subject.lab
           for lab in labs:
              if lab.subject_id == 1:
                 print(lab.name)
                 print(lab.description)
                 print(lab.deadline)
```