

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ТЕЛЕГРАМ БОТА З ВИКОРИСТАННЯМ API ТА SQLITE
DEVELOPMENT OF A TELEGRAM BOT USING API AND SQLITE

Виконав: студент 4 курсу, групи КНТ-218

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

Батюкін І.Д.

(прізвище та ініціали)

Керівник Гладкова О.М.

(прізвище та ініціали)

Рецензент Зеленьова І.Я.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
 Завідувач кафедри Субботін С.О.,
 доктор технічних наук, професор

« » 2022 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Батюкіна Ігоря Денисовича
 (прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка телеграм бота з використанням API та SQLite.
Development of a Telegram Bot Using API and SQLite.

керівник проєкту (роботи) Гладкова Ольга Миколаївна, к.т.н.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “27” квітня 2022 року №102

2. Строк подання студентом проєкту (роботи) червень 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Програмне забезпечення у кальяній індустрії. 2. Використовуване програмне забезпечення для розробки. 3. Розробка Telegram бота.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1,2,3	Гладкова О.М., доцент		
Нормоконтроль	Дейнега Л.Ю., ст.викладач		

7. Дата видачі завдання “04” квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи та підбір літератури.	1 тиждень	Виконано
2	Аналіз предметної області.	2-3 тижні	Виконано
3	Вибір програмного комплексу	4 тиждень	Виконано
4	Вибір апаратного комплексу	5 тиждень	Виконано
5	Налаштування програмних та апаратних засобів	6 тиждень	Виконано
6	Формування висновків та написання пояснювальної записки	7 тиждень	Виконано
7	Захист роботи	8 тиждень	Виконано

Студент

Батюкін І.Д.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи)

Гладкова О.М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
97с., 4 табл., 39 рис., 2 дод., 15 джерел.

TELEGRAM BOT API, БОТ, PYTHON, TELEGRAM, SQLITE 3, SUBLIME
TEXT3.

Об'єкт дослідження – програмні засоби для каляйної індустрії.

Предмет дослідження – бот в Telegram для об'єднання аудиторії, що цікавляться каляйною індустрією.

Мета роботи – створення бота в телеграм для каляйної промисловості з використанням бази даних SQLite.

Матеріали, методи та технічні засоби: функціональне програмування, мови програмування Python, бібліотеки TelegramBotApi, Base64, Uuid4, Requests, Json, LabeledPrice, ShippingOption, ноутбук з процесором Intel Core i7-8750H під управлінням операційної системи Windows.

У роботі проводиться аналіз Telegram api для створення зручного інтерфейсу управління та створення широкого функціоналу.

Покрокове написання коду та реалізація функціоналу.

Клієнтської частини: Купівля/Продаж, Новини, Вакансії, Відгуки та ін.

Інструмент адміністратора: Модерація відгуків, Публікація Новин, Модерація оголошень, Модерація вакансій.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 97 pages, 4 tables, 39 figures, 2 appendixes, 15 sources.

TELEGRAM BOT API, BOT, PYTHON, TELEGRAM, SQLITE 3,
SUBLIME TEXT3.

The object of research is software for the hookah industry.

The subject of the study is a bot in Telegram to unite audiences interested in the hookah industry.

The purpose of the work is to create a bot in telegrams for the hookah industry using the SQLite database.

Materials, methods and technical means: functional programming, Python programming languages, TelegramBotApi libraries, Base64, Uuid4, Requests, Json, LabeledPrice, ShippingOption, laptop with Intel Core i7-8750H processor running Windows operating system.

The paper analyzes the Telegram api to create a user-friendly management interface and create a wide range of functionality.

Step-by-step code writing and implementation of functionality.

Client part: Buy / Sell, News, Vacancies, Reviews, etc.

Admin Tool: Review Moderation, News Publishing, Ad Moderation, Job Moderation.

ЗМІСТ

С.

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	7
ВСТУП	8
1 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ У КАЛЬЯННІЙ ІНДУСТРІЇ.....	9
1.1 Унікальність проєкту.....	9
1.2 Огляд Telegram API	10
1.3 Огляд різновидів Telegram ботів	11
1.4 Висновки за розділом	12
2 ВИКОРИСТОВУВАНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗРОБКИ...	13
2.1 Огляд особливостей мови програмування Python.....	13
2.2 Вибір середовища розробки	15
2.3 Огляд та вибір бази даних для підключення до Telegram	16
2.4 Висновок за розділом	18
3 РОЗРОБКА TELEGRAM БОТА	19
3.1 Загальна структура проєкту	19
3.2 Алгоритми роботи з ботом	19
3.3 Опис розробленого бота та його функціонал	21
3.3.1 Купити	24
3.3.2 Продати	28
3.1.3 Чат, відгуки, вакансії.....	32
3.1.4 Новини.....	33
3.1.5 Підтримка проєкта	35
3.4 Організація бази даних.....	36
ВИСНОВКИ.....	44
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	45
ДОДАТОК А Текст програми.....	47
ДОДАТОК Б Слайди презентації	90

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- API – Application Programming Interface (програмний інтерфейс програми);
- ООП – об'єктоорієнтоване програмування;
- БД – бази даних;
- ID – ідентифікатор, унікальна ознака об'єкту.

ВСТУП

У сучасних реаліях, люди повсякденно використовують смартфони і більше часу вони проводять за соціальними мережами та месенджерами, одним з найпопулярніших месенджерів на даний момент є Telegram.

Він активно розвивається, додаючи нові функції, люди можуть створювати групові чати для спілкування, канали для ведення свого блогу і навіть вести прямі трансляції.

Telegram додали ботів, які автоматизували багато процесів, наприклад:

- модерація чатів;
- автовідповіді на запитання, що часто ставляться;
- платежі між людьми;
- торгівля і багато іншого.

Боти в Telegram стали головною фішкою цієї програми, давши можливість людям робити все в пару кліків не виходячи з програми, не витрачаючи час на переходи на сайт і не відриваючись від листування з друзями. Таким чином робота присвяченню вирішенню актуального завдання.

У цій роботі буде розглянуто створення одного з ботів, на прикладі кальяної індустрії. Унікальність бота буде в простоті інтерфейсу, а так само в поєднанні багатьох аспектів, яких немає в інших.

Для вирішення поставленої мети необхідно виконати наступні завдання:

- дослідити предметну область;
- виконати аналіз процесу розробки боту;
- розробити структуру пояснювальної записки;
- виконати програмну реалізацію проекту;
- провести тестування програмного продукту.

1 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ У КАЛЬЯННІЙ ІНДУСТРІЇ

У нашій країні, кальян відносно новий тренд. У цій сфері завжди була велика аудиторія, але дуже мала кількість людей у неї сильно занурювалися. Наразі ця сфера активно розвивається в нашій країні, за останні роки з'явилося багато вітчизняних виробників кальянів, тютюнових сумішей, чаш та інших аксесуарів.

У результаті кількість людей, які активно цікавляться розвитком цієї сфери, бажають дізнатися про новинки, побільшало.

Тому мені прийшла ідея створити місце в якому зможуть зібратися як професіонали так і любителі.

Я вирішив створити бота де користувач зможе купити або продати свій кальян або інший пов'язаний з цим товар, дізнатися про новини індустрії, знайти роботу в цій сфері, а також просто поспілкуватися з однодумцями.

1.1 Унікальність проєкту

Є безліч додатків створених на тематику кальянів, але основна їх мета міксологія. Вони створені для того, щоб людина могла знайти мікс смаку для себе, на цьому завдання їх функціонал і закінчується. Що не мало, але не охоплює всіх аспектів цієї сфери.

Зараз існує безліч магазинів кальної атрибутики. Найвідоміші вважаються: Tornado Hookah Shop, SkyMarket.

Так само є і кальян барахолка України, але це лише чат в телеграм або публічний аккаунт в Інстаграм.

Тобто єдиний варіант для людини продати свій старий апарат або будь-яку іншу атрибутику це йти на сайти типу OLX.

Проаналізувавши це було вирішено додати у проєкт функцію для публікації товарів на продаж, а також можливість іншим знайти відповідний товар для себе.

1.2 Огляд Telegram API

Telegram використовує свій протокол шифрування MTProto. MTProto API (він же Telegram API) - це API, через який ваша програма Telegram зв'язується з сервером. Telegram API повністю відкритий, тому будь-який розробник може написати свій клієнт месенджера.

Для написання ботів було створено Telegram Bot API — надбудову над Telegram API [1].

Щоб використовувати Bot API, вам не потрібно нічого знати про те, як працює протокол MTProto - наш допоміжний сервер буде сам обробляти все шифрування і зв'язок з Telegram API. Ви з'єднуєтеся з сервером через простий інтерфейс HTTPS, який надає просту версію Telegram API[1].

Серед спрощень Bot API: робота через вебхукі, спрощена розмітка повідомлень та інше[1].

Чомусь мало хто знає, що боти можуть працювати безпосередньо через Telegram API. Більше того, у такий спосіб можна навіть обминути деякі обмеження, які дає Bot API[1].

Найголовніше знати про `user_id` та `message_id`: id користувачів та чатів.

У кожного користувача, бота, групи, каналу в Телеграм є власний id. Розрізняти чати в коді робота слід саме за id, тому що він ніколи не змінюється[1].

У токени робота перша частина - це його id. Наприклад, токен `110201874:AAHdqTcvCH1vGWJxfSeofSAs0K5PALDsaw` належить боту з id `110201874` [1].

У Bot API перед id супергруп і каналів пишеться -100. Так, id 1356415630 перетворюється на -1001356415630.

Не слід зберігати ID користувачів та чатів у 32-бітному типі числа: тепер ID можуть перевищувати 231-1 [1].

Кожне повідомлення у Телеграмі має свій id. Це відноситься і до системних повідомлень (користувач зайшов до групи, змінилася назва групи і т.д.)

Через Telegram API боти можуть отримувати на запит повідомлення в будь-якому чаті за їх id [1].

У супергрупах і каналах id повідомлень є унікальними для чату: перше повідомлення в чаті має номер 1, друге має номер 2 і так далі [1].

У особистих повідомленнях та звичайних групах id повідомлень працюють інакше. Там, можна сказати, нумерація наскрізна: id повідомлення унікальне для кожного користувача, що його відправив. Так, перше повідомлення від користувача в усіх особи і групах має номер 1, друге повідомлення від того ж користувача має номер 2 і так далі [1].

1.3 Огляд різновидів Telegram ботів

Telegram – кросплатформова система миттєвого обміну повідомленнями (месенджер) з функціями VoIP, що дозволяє обмінюватися текстовими, голосовими та відеоповідомленнями, стікерами та фотографіями, файлами багатьох форматів [2].

Створив його Павло Дуров після відходу з Вконтакте, його метою було створення безпечного месенджера, де кожне повідомлення проходить через унікальну систему шифрування, щоб будова особи не могла отримати доступ до персонального листування інших користувачів.

Ця система стала однією з головних переваг Телеграм, яка і привабила велику аудиторію.

Кількість щомісячних активних користувачів сервісу станом на січень 2021 становить близько 500 млн осіб [2].

Приблизно 1,5 млн. осіб реєструються в Telegram щодня [3].

Телеграм-бот – це спеціальний обліковий запис, створений в автоматичному режимі, який дозволяє користувачам здійснювати різні дії через сам месенджер [3].

Існують різні види ботів:

- магазин – цифрова вітрина товарів;
- функціональний – чудово виконує одну чи кілька функцій, наприклад бот контролю підписок, бот скачування відео чи парсингу сторонніх ресурсів;
- криптовалютний – різні гаманці, обмінники та багато іншого;
- доставка – дозволяє створювати посилки та відстежувати;
- консультанти – відповідає на запитання користувачів із заготовленим скриптом або розсилає інформацію для користувачів.

Всі вони автоматизують різні види діяльності у повсякденному житті людей, допомагаючи швидко виконувати різні повсякденні завдання.

1.4 Висновки за розділом

Під час аналізу було визначено, що розробники приділяють малу увагу до кальяної індустрії, що дає можливість для створення унікального продукту. А Телеграм має зручний інтерфейс розробки та великий майданчик із великою аудиторією для розміщення свого проекту. Таким чином було вирішено реалізувати робота в месенджері телеграм для кальяної індустрії.

2 ВИКОРИСТОВУВАНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗРОБКИ

2.1 Огляд особливостей мови програмування Python

Python – високорівнева мова програмування загального призначення з динамічною строгою типізацією та автоматичним керуванням пам'яттю. Переваги та недоліки цієї мови програмування зведені у таблиці 2.1.

Він орієнтується на легку читабельність коду, підвищення продуктивності розробника. Сама мова повністю об'єкто-орієнтована, що означає, що все є об'єктом [4].

Python з'явився на світ у 1990, але досі не втрачає своєї актуальності [4]. Люди постійно створюють нові бібліотеки для полегшення роботи та створення нових функцій. Сама мова також постійно оновлюється, актуальною версією є Python 3.10.4. Вона вийшла 24 березня 2022 року і саме вона використовувалася у моїй дипломній роботі.

Як писалося раніше, Python використовується в ООП, що означає, що всі об'єкти є похідними класів і успадковують від них атрибути. У цьому кожен об'єкт формує власний простір імен. Python підтримує такі ключові особливості об'єктно-орієнтованого програмування, як успадкування, інкапсуляцію та поліморфізм. Однак інкапсуляцію в розумінні приховування даних Python підтримує лише в рамках угоди, але не синтаксису мови [5].

ООП дає можливість використовувати один код із різними даними. За допомогою класів створюється безліч об'єктів, кожен з яких може мати власні значення полів. Немає необхідності вводити безліч змінних, тому що об'єкти одержують у своє розпорядження індивідуальні простори імен. У цьому сенсі об'єкти схожі структури даних. Об'єкт можна як якусь упаковку даних, до якої приєднані інструменти їхньої обробки – методи [5].

У таблиці 2.1 наведено головні переваги та недоліки мови Python.

Таблиця 2.1 – Переваги та недоліки мови програмування Python

Переваги	Недоліки
Python - це кроссплатформенний, підходить для різних платформ[6].	Є реалізація інтерпретаторів для мобільних пристроїв та непопулярних систем. Весь процес робить його повільно порівняно з компіляцією низького рівня C безпосередньо до машинобудівного коду[7].
Основним плюсом Python є легкий поріг входження, основи цієї мови може освоїти і 7-8 річна дитина[6]. Спрощений синтаксис та інтуїтивно зрозумілі функції дозволяють легко розібратися в основах програмування[6].	Пам'ять Витрати. Типи даних Python є гнучкими, що викликає деякі вищі накладні витрати[7].
Має велику аудиторію, що полегшує роботу з ним, тому що є багато матеріалів та форумів для пошуку вирішення питань[6].	Мобільний розвиток: Хоча можна розробити мобільні програми в Python, це не основна увага пітона. Мова програмування, така як Java, підійде вашим потребам краще: більшість людей сьогодні розроблять програми для Android в Java[7].
А також багато великих компаній, підтримують Python, при написанні Арі до своїх проектів вони передбачають використання розробниками Python[6].	Рідний GPGPU для навчання машини: Python – це надто багато рівня програмування високого рівня для обробки високої паралелістичної та дрібнозернистої пам'яті контролю процесора HPU[7].
Є реалізація інтерпретаторів для мобільних пристроїв та непопулярних систем[6].	

Порівняно з багатьма іншими мовами у Python об'єктно-орієнтоване програмування має низку особливих рис.

Все є об'єктом – число, рядок, список, функція, екземпляр класу, клас, модуль. Так клас – об'єкт, здатний породжувати інші об'єкти – екземпляри. У Python немає просто типів даних. Усі типи – це класи [5].

Інкапсуляція в Python не приділяється особливої уваги. В інших мовах програмування зазвичай не можна отримати безпосередньо доступ до якості, описаної в класі. Для його зміни може бути передбачено спеціальний метод. У Python ж не вважається поганим безпосереднє звернення до властивостей [5].

Крім інкапсуляції, спадщини та полеморфізму у Python є ще один - більш елегантний спосіб визначення властивостей. Цей спосіб передбачає використання анотацій [8].

2.2 Вибір середовища розробки

За допомогою IDE або навіть хорошого редактора коду можна спростити собі життя. Але питання – що обрати [9].

В таблиці 2.2 наводиться порівняння найрозповсюджених середовищ розробки для мови Python.

Таблиця 2.2 – Порівняння середовищ розробки

Назва	Переваги	Недоліки
Eclipse + PyDev	Якщо у вас вже було встановлено Eclipse, то установка PyDev пройде швидко та гладко. У досвідченого користувача Eclipse не виникне проблем із вивченням цього розширення[9].	Якщо ви починаєте вивчати Python або розробку в цілому, Eclipse може стати непосильною ношею[9]. Складний у налаштуванні.

Продовження таблиці 2.2

Назва	Переваги	Недоліки
Sublime Text 3	Швидкий, легкий та має гарну підтримку. Мінімалістичний дизайн, підтримка різних операційних систем[9].	Sublime Text не є безкоштовним, хоча ви можете використовувати пробний період скільки завгодно. Установка розширень може перетворитися на ще квест. Крім того, у редакторі немає підтримки налагодження та запуску коду[9].
Atom	Підтримка на всіх платформах завдяки Electron. Atom легко і швидко скачується та завантажується[9].	Підтримка збирання та налагодження не вбудована, а додається за допомогою розширень. Оскільки Atom написаний за допомогою Electron, він завжди працює як JavaScript-процес, а не як нативна програма[9].

Писати на Python, використовуючи IDLE або Python Shell, цілком зручно, якщо йдеться про прості речі, проте зі зростанням розмірів проєктів програмування перетворюється на муку.

2.3 Огляд та вибір бази даних для підключення до Telegram

Якщо раніше вже використовувалась якась БД, можна просто сказати: «Я виберу цю базу, тому що знайомий з нею». Це цілком підходяще рішення, коли продуктивність не є критичною для вашого проєкту. В іншому випадку вибір невідповідної бази стане перешкодою для розширення проєкту. Виправити помилку може бути досить складно. Навіть якщо ви працюєте у зрілому проєкті, який використовує конкретну БД, важливо знати її обмеження та розуміти, коли

варто додати базу іншого типу до вашого стеку. Комбінування кількох баз даних досить поширене [10]. Порівняння БД наведено в таблиці 2.3.

Таблиця 2.3 – Порівняння баз даних

Назва	Переваги	Недоліки
MongoDB	Схема менша. Якщо у вас є гнучка схема, це ідеально підходить для сховища документів. Легкість масштабування. Масштабування читається з допомогою наборів реплік. Масштабування записується за допомогою функції sharding. MongoDB є безкоштовним і може працювати на Linux[11].	Розмір даних у MongoDB зазвичай вищий через, наприклад, кожен документ має імена полів, які зберігаються в ньому. Менше гнучкості при запиті (наприклад, немає JOIN). Немає підтримки транзакцій[11].
PostgreSQL	Легко використати. Має тип даних користувача. Відкрите джерело. Велика підтримка спільноти. Використовуйте процедури, що зберігаються[12].	Створює окремий сервіс для кожного клієнта, який перетворюється на велике використання пам'яті. Не дуже гарний у плані продуктивності. Робити реплікацію складніше. Встановлення не легке для новачка[12].
SQLite 3	Висока швидкість. Збереження даних в одному файлі. Мінімалізм. Надійність. Код на 100% покритий тестами. Це означає, що протестований кожен компонент ПЗ.Нульова конфігурація. Перед використанням СУБД не потрібне складне налаштування або тривале встановлення. Кросплатформність[13].	Обмежена підтримка типів даних. Відсутність процедур, що зберігаються. Обмеження у застосуванні. Відсутність вбудованої підтримки Unicode[13].

2.4 Висновок за розділом

Виходячи з таблиць та вивчених мною матеріалів, я вибрав Python для написання коду. Так як у Telegram Api, є підтримка цієї мови, він прос у використанні і локанічний. Також є багато документації по роботі з Telegram Bot Api і Python.

Середу написання коду я вирішив використати Sublime Text 3. Надійне середовище розробки, без зайвих функцій, всі необхідні плагіни я можу встановити додатково. Наприклад Anaconda – дозволяє виправляти код під автоформат PEP 8 [14].

Вибір бази даних, тут я відразу зупинився на SQLite 3. Він чудово підходять для моїх задач. Інтерфейс дозволяє легко орієнтуватися в БД.

3 РОЗРОБКА TELEGRAM БОТА

3.1 Загальна структура проєкту

На рисунку 3.1 зображена загальна схема структури проєкту.

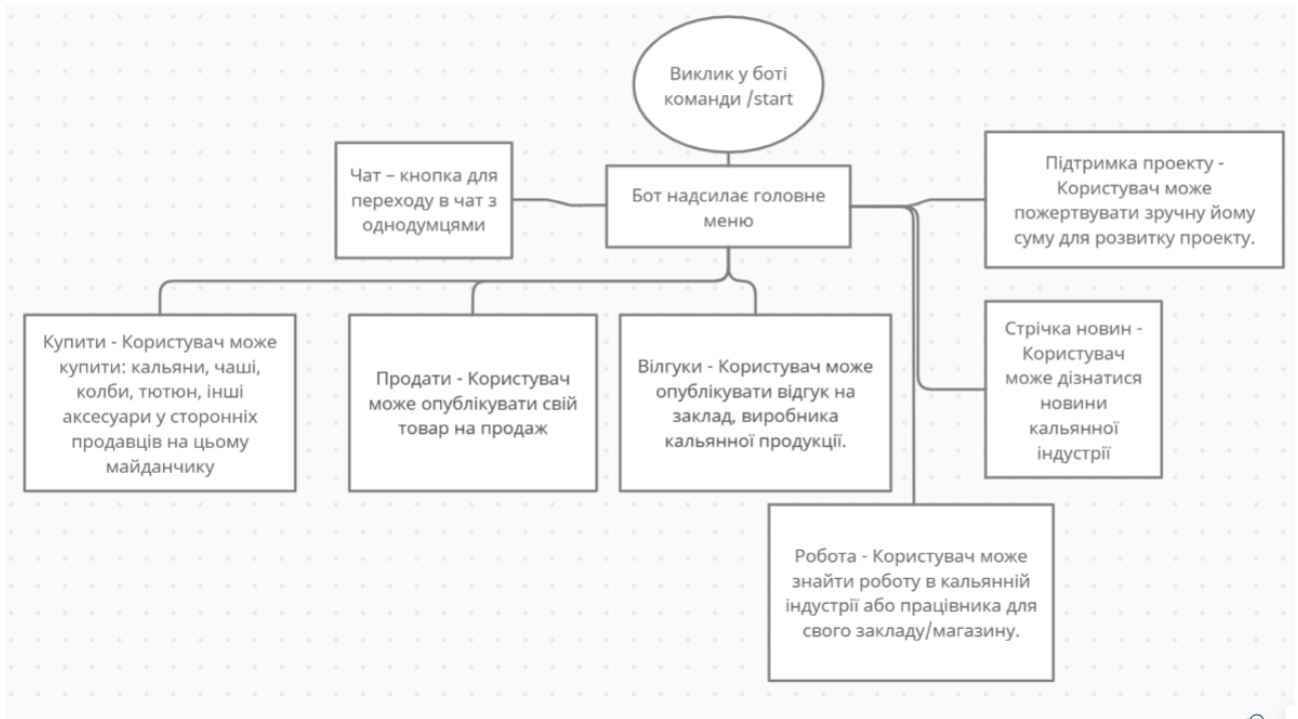


Рисунок 3.1 – Структура проєкту

3.2 Алгоритми роботи з ботом

На основі розробленої структури проєкту було побудовано схему алгоритму покупки зображено на рисунку 3.2. На рисунку 3.3 схема алгоритму продажу.

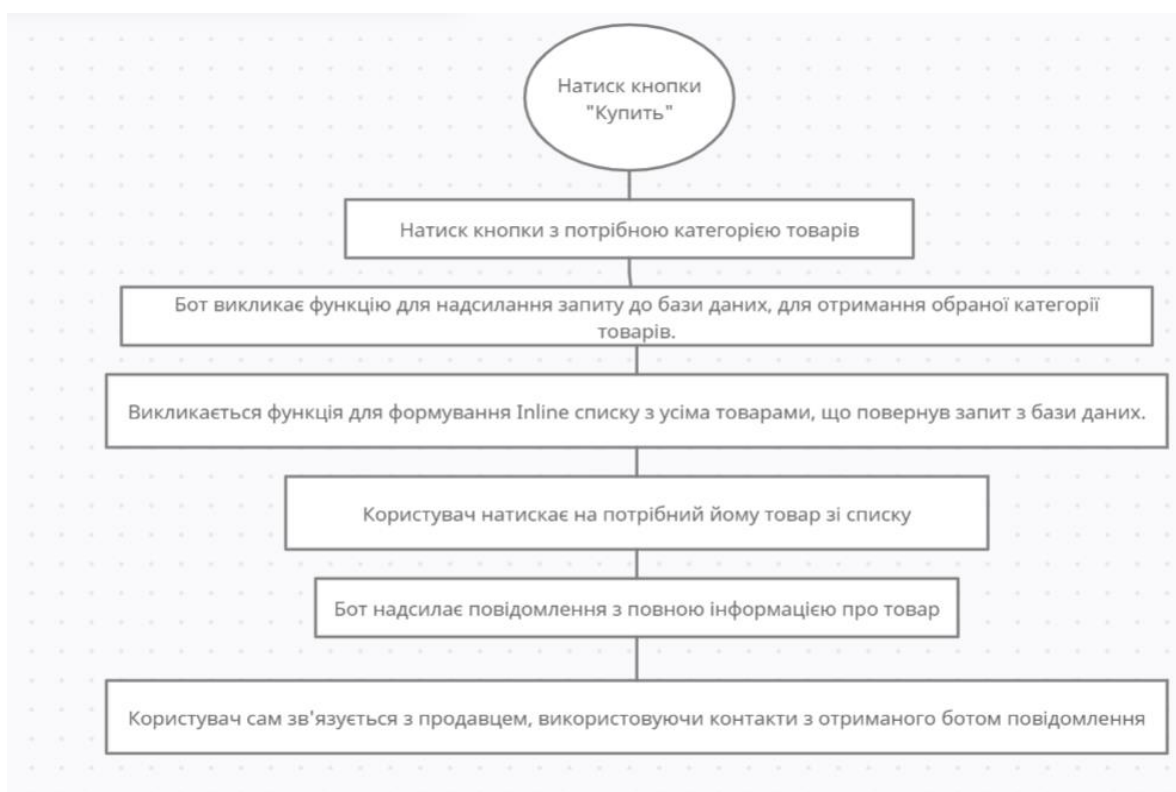


Рисунок 3.2 – Алгоритм покупки

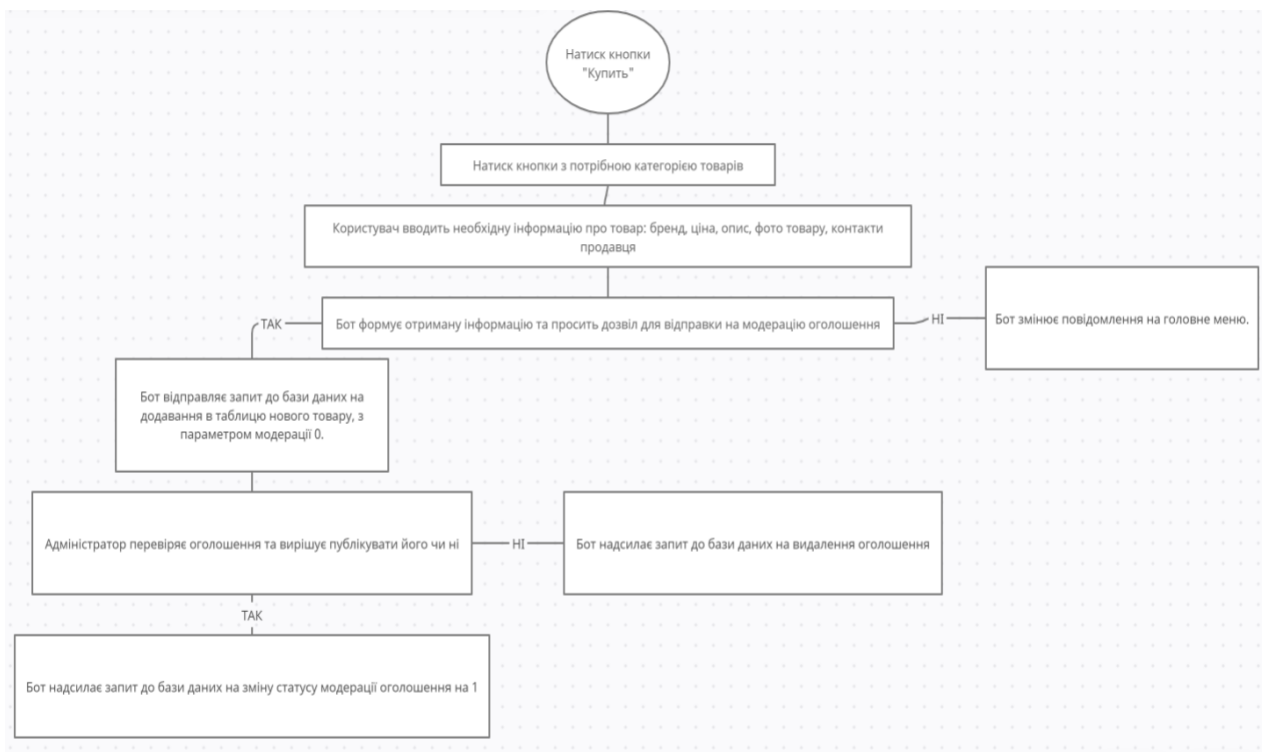


Рисунок 3.3 – Алгоритм продажу

На рисунку 3.4 наводиться алгоритму пошуку або розміщення вакансій відповідно.

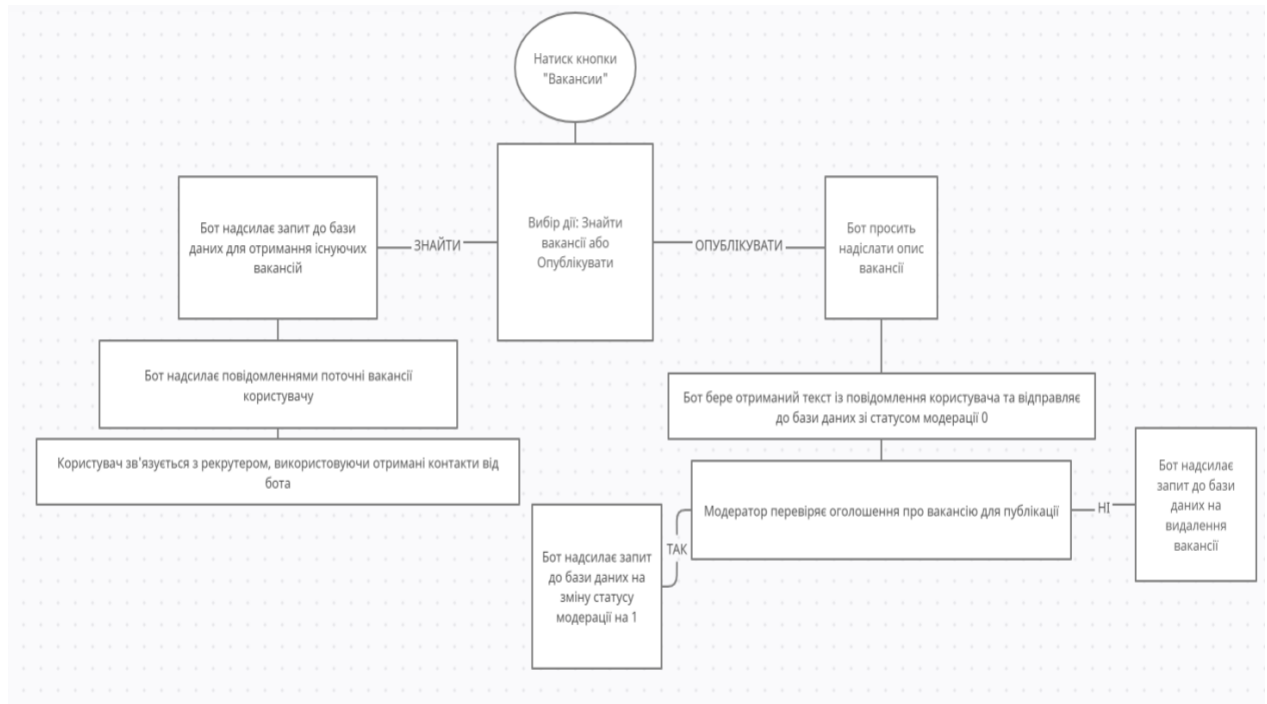


Рисунок 3.4 – Алгоритм вакансії

3.3 Опис розробленого бота та його функціонал

Перше, що відбувається, коли користувач заходить в будь-якого бота це відпрацювання команди /start рис. 3.5.

```

@bot.message_handler(commands=['start'])
def message_handler(message):
    cid = message.chat.id
    db_cmd.check_user_id(cid)
    bot.send_message(message.chat.id, "Приветствуем в HookahHome.\n"+
    "Здесь мы собрали все что может быть вам интересно по кальяной индустрии.\nДля обратной связи @tedillumDevTelegam.",
    reply_markup=markup.gen_mainmenu())
  
```

Рисунок 3.5 – Код обробки команди /start

Перше, що відбувається, коли користувач заходить в будь-якого бота це виклик команди /start. У коді у мене є обробник цієї команди.

@bot.message_handler – перехоплює повідомлення від користувача.

У разі заданий тип повідомлення це команда start.

Тобто кожен раз, коли користувач буде відправляти в бота повідомлення з текстом "/start" в моєму коді відбуватиметься подія перехоплення повідомлення, після чого викликається функція message_handler, яку я написав сам.

У ній я перехоплюю повідомлення, щоб отримати cid - унікальний ідентифікатор чату, кожен користувача при спілкуванні з ботом він є. Він допомагає визначити боту в якій чат, що потрібно відповісти.

Після чого я перевіряю функцією check_user_id чи є даний користувач у базі даних. Якщо ні, то відбувається виклик функції add_user, щоб додати його до бази даних рис. 3.6.

```

71 def check_user_id(user_id):
72     db_file = "db.db"
73     conn = None
74     check = False
75     try:
76         sql = f"SELECT * FROM users WHERE user_id = {user_id}"
77         conn = sqlite3.connect(db_file)
78         cur = conn.cursor()
79         cur.execute(sql)
80         data = cur.fetchone()
81         if not data:
82             cur.close()
83             add_user(user_id)
84             check = True
85         else:
86             cur.close()
87     except Exception as e:
88         print(e, 1)
89     finally:
90         if conn is not None:
91             conn.close()
92     return check
93
94 def add_user(user_id):
95     db_file = "db.db"
96     conn = None
97     try:
98         sql = "INSERT INTO users(user_id, state, data,mid) VALUES(?, ?, ?, ?)"
99         conn = sqlite3.connect(db_file)
100        cur = conn.cursor()
101        mass = {'category': '', 'brand': '', 'price': '', 'discrip': '', 'contact': '', 'photo': ''}
102        cur.execute(sql, (user_id, 0, str(mass), 0))
103        conn.commit()
104        cur.close()
105    except Exception as e:
106        print(e, 1)
107    finally:
108        if conn is not None:
109            conn.close()

```

Рисунок 3.6 – Код перевірки користувача

Після перевірки користувача, я викликаю функцію `bot.send_message()`, яка вбудована в Арі Telegram, щоб бот надіслав повідомлення користувачу, з привітання та стартовим меню рис. 3.7.

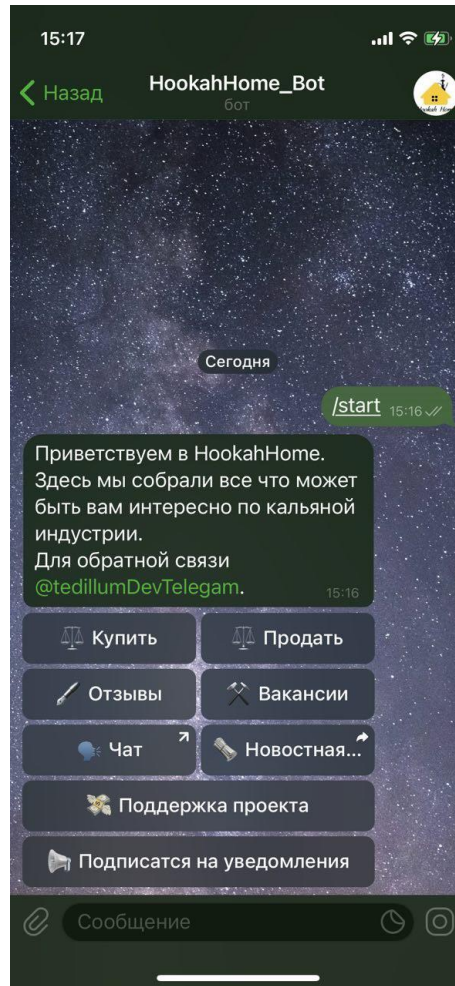


Рисунок 3.7 – Стартовое меню

Меню формується у файлі `markup.py`, де створено функцію `def gen_mainmenu()`.

У ній створюються десять кнопок та організовуються в один список `markup`, який функція повертає при виклику функції рис. 3.8.

```
def gen_mainmenu():
    markup = types.InlineKeyboardMarkup(row_width=2)
    btn1 = types.InlineKeyboardButton(
        "🛒 Купить", callback_data="buy")
    btn10 = types.InlineKeyboardButton(
        "🛒 Продать", callback_data="sall")
    btn2 = types.InlineKeyboardButton(
        "📝 Отзывы", callback_data="reviews")
    btn3 = types.InlineKeyboardButton(
        "🔗 Вакансии", callback_data="work")
    btn4 = types.InlineKeyboardButton(
        "📰 Новостная лента", switch_inline_query_current_chat="news")
    btn6 = types.InlineKeyboardButton(
        "💬 Чат", url='https://t.me/joinchat/7mdmivmckJg00DUy')
    btn5 = types.InlineKeyboardButton(
        "🛠 Поддержка проекта", callback_data="help")
    btn9 = types.InlineKeyboardButton(
        "🔔 Подписаться на уведомления ", callback_data="scrib")
    markup.add(btn1, btn10)
    markup.add(btn2, btn3)
    markup.row(btn6, btn4)
    markup.row(btn5)
    markup.row(btn9)
    return markup
```

Рисунок 3.8 – Організація меню

3.3.1 Купити

При натисканні кнопки "Купити" спрацьовує перехоплювач для кнопки `@bot.callback_query_handler(func=lambda call: call.data == "buy")`

Де `call.data` – дані кнопки з її назвою, `==` – оператор порівняння рис. 3.9.

```
#-----BUY-----
@bot.callback_query_handler(func=lambda call: call.data == "buy")
def callback_buy(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            bot.edit_message_text(
                chat_id=cid,
                message_id=mid,
                text=" Выберите товар ",
                parse_mode="Markdown",
                reply_markup=markup.categor())
    except Exception as e:
        print('callback_buy:', e)
```

Рисунок 3.9 – Кнопка «Купить»

Спочатку я беру ідентифікатори чату, користувача та повідомлення.

Я викликаю функцію `bot.edit_message_text` для того, щоб змінити повідомлення, до якого була прив'язана кнопка. Для оптимізації інтерфейсу, я змінюю текст і викликаю наступне меню з категоріями товарів рис. 3.10. На рисунку 3.11 зображено меню категорії товару.

```
def categor():
    markup = types.InlineKeyboardMarkup(row_width=2)
    btn1 = types.InlineKeyboardButton(
        "Кальяны", switch_inline_query_current_chat="hookah/")
    btn2 = types.InlineKeyboardButton(
        "Чаши", switch_inline_query_current_chat="bowls/")
    btn3 = types.InlineKeyboardButton(
        "Табак", switch_inline_query_current_chat="tobac/")
    btn4 = types.InlineKeyboardButton(
        "Колбы", switch_inline_query_current_chat="colbs/")
    btn5 = types.InlineKeyboardButton(
        "Акссесуары", switch_inline_query_current_chat="other/")
    btn6 = types.InlineKeyboardButton(
        "Главное меню", switch_inline_query_current_chat="start")
    markup.add(btn1, btn2)
    markup.add(btn3, btn4)
    markup.row(btn5)
    markup.row(btn6)
    return markup
```

Рисунок 3.10 – Меню Категорії

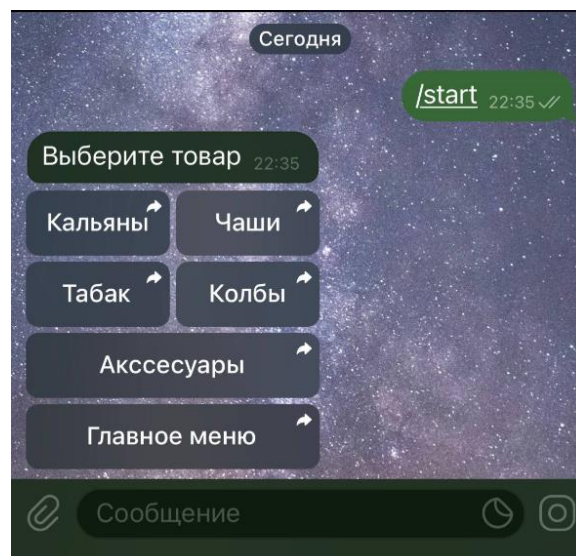


Рисунок 3.11 – Меню Категорії

Тепер я не використовую кнопки `callback_data`, а створюю їм `switch_inline_query_current_chat`. Це зміна необхідно, щоб використовувати Inline Mode, за допомогою нього я оптимізую інтерфейс товарів, щоб не надсилати багато повідомлень з кожним товаром, я вирішив використовувати цей мод, який вбудований в Telegram[17]. Так я організую всі товари обраної категорії до списку, який видасться у вигляді стрічки, яку можна перегорнути, відсортувати за назвою чи ціною рис. 3.12.

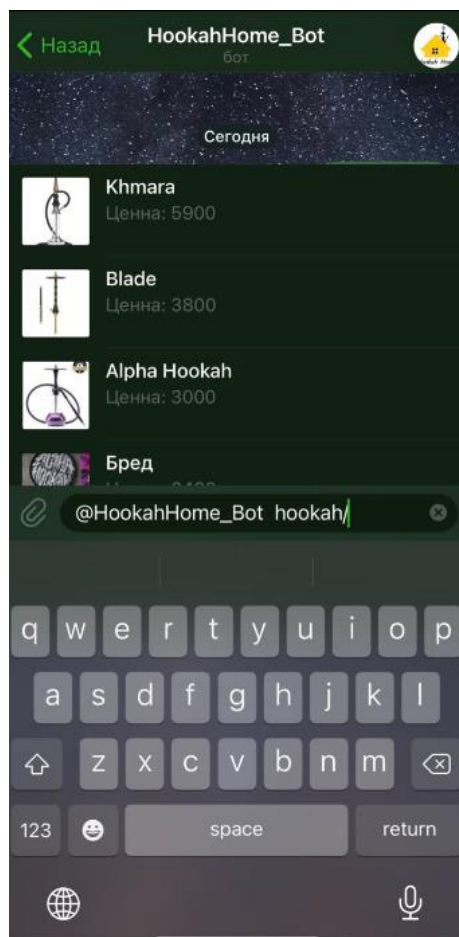


Рисунок 3.12 – Меню Товарів

Коли користувач з цього списку вибере товар, відпрацює подія `@bot.inline_handler(lambda query: types.InlineQuery)`, яка дасть мені інформацію

про те, який товар був обраний[17]. І я видам повну інформацію про нього рис. 3.13.



Рисунок 3.13 – Опис товару

Спочатку викликається функція `db_cmd.get_product_inline(inline_query.query.split('/')[0])`, в яку передається назва категорії, яку вибрав користувач, щоб отримати список товарів цієї категорії.

Далі йдуть кілька `if` - оператор для перевірки умови, якщо вона виконується, виконує код всередині нього.

Тут `inline_query.query.split('/')[1]==''`, якщо запит користувача не змінюється, то видає всі товари.

Тут відбувається сортування за ціною
`inline_query.query.split('/')[1]).isdigit()==True` and
`i[4]>=int(inline_query.query.split('/')[1])`, якщо після hookah/ пишеться цифри, то
 видаються товари починаючи від цієї ціни.

Сортування за назвою: `(inline_query.query.split('/')[1]).isdigit()==False` and
`i[3].count(inline_query.query.split('/')[1])>0` .

І тепер я видаю товари за допомогою
`bot.answer_inline_query(inline_query.id, list_inline,0,switch_pm_parameter="start")`.

А якщо товар зацікавив, там обов'язково буде вказано контакти продавця
 для зв'язку.

3.3.2 Продати

У цьому пункті меню я додав можливість користувачеві додавати
 оголошення для продажу.

Так, від користувача при натисканні кнопки бот запросить ввести певні
 перелік інформації про товар:

- вибір категорії товару;
- назва бренду товару;
- ціну товару;
- короткий опис, це може бути характеристики, якість товару і таке
інше;
- контактні дані продавця тут можна вказати як і свій телеграм для
спрощення зв'язку, так і просто номер телефону;
- зображення, щоб інші ользователи могли візуально оцінити товар.

Після того, як людина ввела інформацію про товар, її запитують чи
 надсилати оголошення на модерацію рис. 3.14.

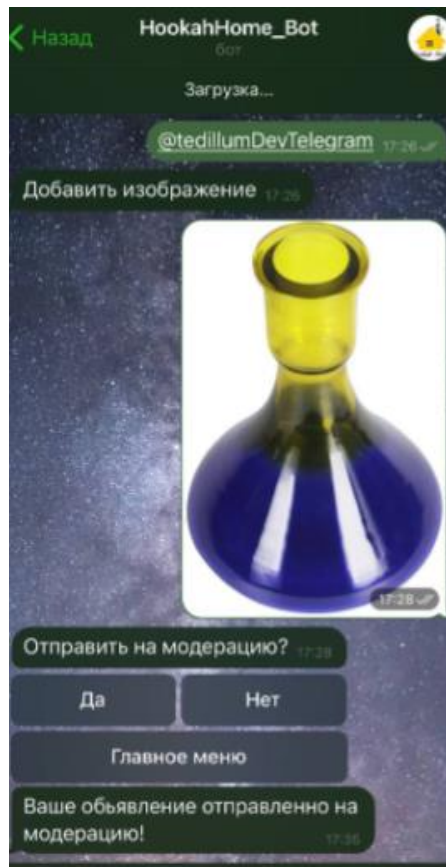


Рисунок 3.14 – Публікація товару

У кодї будь-яка дія користувача змінює його статус, який зберігається в базі даних.

Тому, коли користувач що-небудь відправляє в робота, я перевіряю його статус, щоб не сталося не передбачених наслідків. У цьому мені допомагає оператор "if-elif-else", він відстежує статус користувача і порівнює, якщо він збігається, то виконує задані дії рис. 3.15.

Наприклад, спочатку буде статус "add_brand", значить виконається "if" на 492 рядку.

Відповідно буде виконуватись наступний код, я дістаю параметр data з бази даних, в нього ми записуватимемо ті дані, що ввів користувач про товар.

Виглядає цей параметр як тип даних dict, де є ключ і значення, закріплене за цим ключем.

```

492 if state_user == 'add_brand':
493     data = db_cmd.get_data(uid)
494     d2 = dict(ast.literal_eval(data))
495     print(message.text)
496     d2["brand"] = message.text
497     print(d2)
498     db_cmd.update_user_data(str(uid), str(d2), 'add_price')
499     bot.send_message(
500         cid,
501         text=" Введите цену вашего товара ",
502         parse_mode="Markdown",
503         reply_markup=markup.set_news())
504 elif state_user == 'add_price':
505     data = db_cmd.get_data(uid)
506     d2 = dict(ast.literal_eval(data))
507     if (message.text).isdigit() == True:
508         d2["price"] = message.text
509         db_cmd.update_user_data(str(uid), str(d2), 'add_dscr')
510         bot.send_message(
511             cid,
512             text=" Введите описание о товаре(Высота,материал,состояние и тд) ",
513             parse_mode="Markdown",
514             reply_markup=markup.set_news())
515     else:
516         bot.send_message(
517             cid,
518             text=" Введите цену числами! ",
519             parse_mode="Markdown",
520             reply_markup=markup.set_news())

```

Рисунок 3.15 – Публікація товару

Тому ми за ключем "brand" - буде закріплюватися текст повідомлення, яке ми перехопили.

І наприкінці, пишемо що бот відправив повідомлення з проханням вказати ціну товару.

Якщо користувач погоджується відправляти на модерауцію, бот очищає всі повідомлення і викликає головне меню, зроблено це для того, щоб не перевантажувати інтерфейс користувача.

Оголошення чекає на підтвердження від даміністратора, що робиться в адмін-панелі рис. 3.16.

Це обов'язковий процес, щоб уникнути неприємних наслідків із оголошеннями не доречного характеру.

Модерація відбувається вручну, адміністратор сам перевіряє оголошення та приймає рішення про її публікацію рис. 3.17.

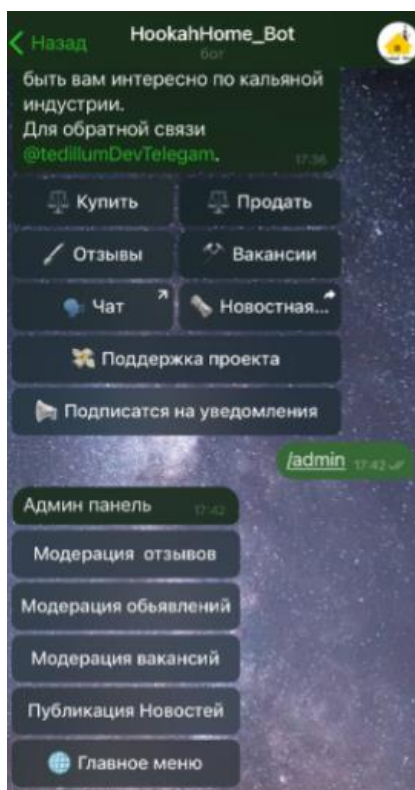


Рисунок 3.16 – Адмін панель

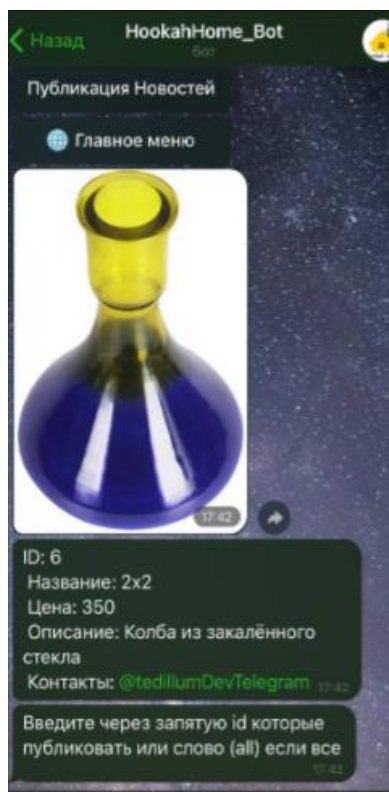


Рисунок 3.17 – Мацерация объявления

3.1.3 Чат, відгуки, вакансії

Це три найпростіші пункти в даному боті, кнопка "Чат", просто переводить в чат телеграма користувача, за допомогою заздалегідь записаної в кнопку посилання на чат.

Вона не вимагає спеціальної обробки та функцій, все передбачено Арі
рис. 3.18.

```
btn6 = types.InlineKeyboardButton(
    "🗨 Чат", url='https://t.me/joinchat/7mdmivmckJg00DUy')
```

Рисунок 3.18 – Кнопка Чат

Після натискання кнопки "Вакансії" або "Відгуки" буде запропоновано ще 2 кнопки рис. 3.19:

- а) для перегляду чужих вакансій/відгуків;
- б) для додавання своїх.

Всі відгуки та вакансії також проходять модерацію, так само як і оголошення. При натисканні цих кнопок, відбувається перехоплення події і бот змінює статус користувача в базі даних, на який відповідає дії користувача.

Це необхідно, щоб далі розуміти в яку таблицю заносити дані.

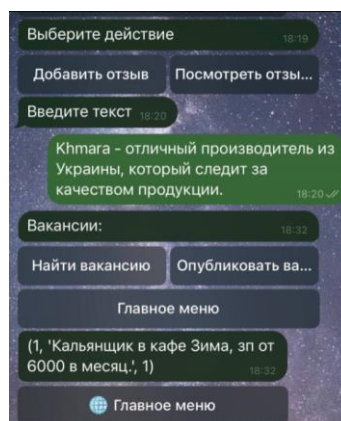


Рисунок 3.19 – Відгуки та вакансії

3.1.4 Новини

Кнопка новини, має на увазі, що висвітлюватимуться новини кальяної індустрії. Стрічку новин наповнюватимуть адміністратори.

Для користувачів новини будуть відображатися так само, як і товари для продажу.

Так як новин може бути багато і для зручного скролінгу краще використовувати InLine Mode рис. 3.20.

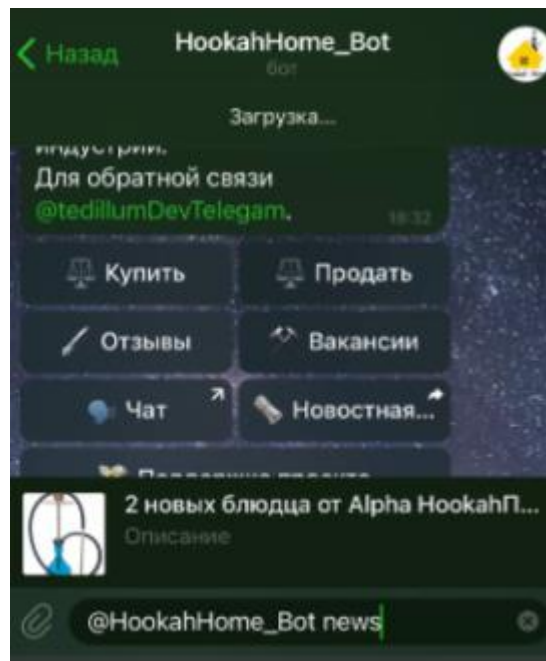


Рисунок 3.20 – Перелік новин

Код додавання новин, дуже простий я отримую текст для новини, потім змінюю статус адміністратора на отримання від нього зображення для новини.

Далі, коли він надсилає зображення, в коді працює перехоплювач для зображень `@bot.message_handler(content_types=['photo'])`.

У ньому оператор `if` визначає статус адміністратора та в залежності від статусу виконує потрібну ділянку коду.

У якому код зберігає id-зображення, тут я уточню.

Коли користувач відправляє якийсь файл у бота, вони зберігаються і кешуються на серверах Telegram, після чого у них є унікальний id, яким можна до них звертатися. Даний спосіб полегшує зберігання великих обсягів інформації, так як ми не перевантажуємо сервер.

На рисунку 3.21 зображена реалізація у коді.

```
@bot.message_handler(func=lambda m: True)
def echo_all(message):
    try:
        cid = message.chat.id
        uid = message.from_user.id
        mid = message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            state = data = db_cmd.get_state_users_admin(uid)
            state_user = db_cmd.get_state_users(uid)
            print(state)
            print(state_user)
            if state == 'add_text_news':
                text_news = message.text
                db_cmd.up_text_users_admin(text_news, uid)
                db_cmd.up_state_users_admin('add_img_new', uid)
                bot.send_message(
                    chat_id=cid,
                    text="Добавить изображение ",
                    parse_mode="Markdown",
                    reply_markup=markup.set_img())
```

Рисунок 3.21 – Код новин

На рисунку 3.22 зображена реалізація у коді.

```
if state == 'add_img_new':
    file_info = bot.get_file(
        message.photo[len(message.photo) - 1].file_id)
    img = file_info.file_id
    text_news = db_cmd.get_text_admin(uid)
    db_cmd.set_news(text_news, img)
    db_cmd.up_state_users_admin(0, uid)
    bot.send_message(
        chat_id=cid,
        text="Новочть добавлена ",
        parse_mode="Markdown",
        reply_markup=markup.gen_adminmenu())
```

Рисунок 3.22 – Код зображення новин

3.1.5 Підтримка проєкта

Так як проєкт спочатку, не планується як комерційний, прибуток приносить хіба що рекламні пости, які пізніше будуть додані в проєкт. Я вирішив додати кнопку для збору донатів, для підтримки проєкту його ж користувачами.

Для реалізації платіжних систем в телеграмах існує стандартний код, який використовує бібліотеки LabeledPrice, ShippingOption[16].

На рисунку 3.23 зображена, як це виглядає в боті, а на рисунку 3.24 зображена реалізація у коді.

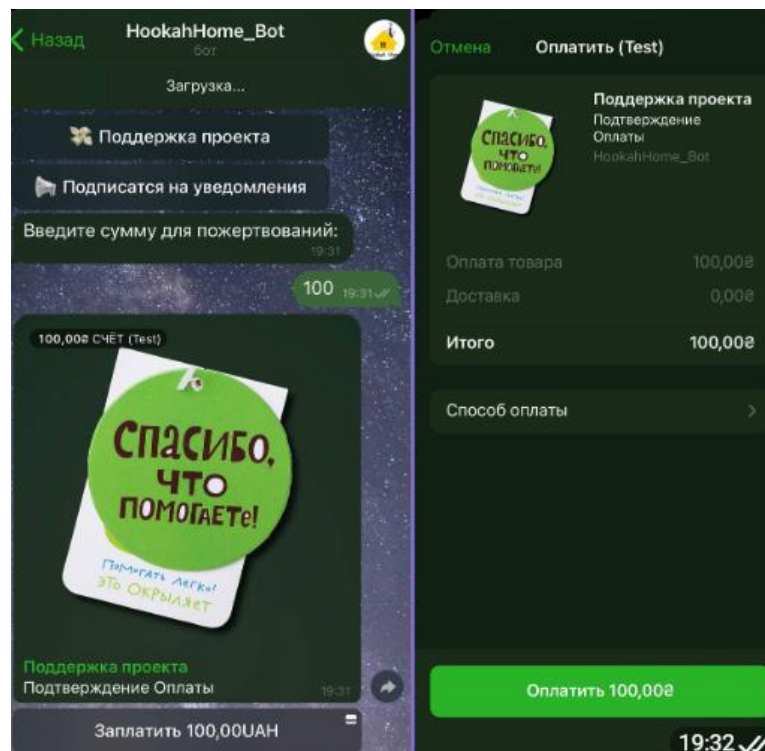


Рисунок 3.23 – Підтримка

```
@bot.callback_query_handler(func=lambda call: call.data == "help")
def callback_pay(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        db_cmd.update_user_data(uid, {'category': 'tobac', 'brand': '', 'price': '', 'discrip': '', 'contact': '', 'photo': ''}, 'pay_for_prog')
        bot.send_message(cid, "Введите сумму для пожертвований:")
    except Exception as e:
        print('callback_pay:', e)
```


В таблиці 3.1 наводиться всі розроблені таблиці бази даних та їх короткий опис, що вони зберігають.

Таблиця 3.1 – Таблиці розробленої БД

Назва таблиці	Опис таблиці
users	Зберігає в собі інформацію про всіх користувачів бота
product	Зберігає в собі інформацію про товари які виставлені на продаж або чекають на модерацію
users_admin	Зберігає в собі інформацію про адміністрацію робота
news	Зберігає інформацію про новини, які опубліковані
reviews	Зберігає у собі всі відгуки, які опубліковані або чекають на модерацію
work	Зберігає у собі всі вакансії, які опубліковані або чекають на модерацію

Усього знадобилося створити 7 таблиць.

Таблиця users має такі поля:

- user_id - поле зберігає у собі id кожного користувача, який використовував бота;
- state - стан користувача, необхідний для відстеження дій користувача, щоб уникнути порушення ланцюжка дій;
- data - параметр для зберігання тимчасових даних, які вводить користувач, наприклад для запалення оголошення про товар;
- text_review – поле для зберігання тексту відгуків або вакансії від користувача;

- mid – поле для зберігання id останнього повідомлення користувача.

На рисунку 3.26 зображено код створення таблиці.

```
def create_tables():
    db_file = "db.db"
    conn = sqlite3.connect(db_file)
    cur = conn.cursor()
    with conn:
        cur.execute(f"""CREATE TABLE IF NOT EXISTS users (
                                user_id INTEGER PRIMARY KEY NOT NULL,
                                state INTEGER NOT NULL,
                                data TEXT NOT NULL,
                                text_review TEXT,
                                mid INTEGER
                            )""")
```

Рисунок 3.26 – Код таблиці users

Код таблиці product зображено на рисунку 3.27 та вона має такі поля:

- id-поле зберігає у собі id товарів, спрощує подальше використання даних;
- user_id - ID користувача, який додав оголошення;
- categor - категорія товару, необхідно для спрощення сортування;
- brand - зберігає у собі назву товару;
- price - ціна товару;
- discrip - опис товару, для додаткової інформації про нього;
- contact - контактні дані продавця для зв'язку;
- photo - фото товару;
- moder - параметр модерації, дозволений він для публікації чи ні.

```

db_file = "db.db"
conn = sqlite3.connect(db_file)
cur = conn.cursor()
with conn:
    cur.execute(f"""CREATE TABLE IF NOT EXISTS product (
        id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE NOT NULL,
        user_id INTEGER NOT NULL,
        categor TEXT NOT NULL,
        brand TEXT,
        price INTEGER NOT NULL,
        discrip TEXT,
        contact TEXT NOT NULL,
        photo TEXT,
        moder INTEGER NOT NULL
    )""")

```

Рисунок 3.27 – Код таблиці products

Код створення таблиці users_admin зображено на рис. 3.28, має такі поля:

- id - поле зберігає в собі id адміністратора;
- user_id - ID користувача в телеграм, для перевірки доступу до адмін панелі;
- state - зберігає стан адміністратора, для правильного порядку виконання;
- data - зберігає тимчасову інформацію.

```

32 db_file = "db.db"
33 conn = sqlite3.connect(db_file)
34 cur = conn.cursor()
35 with conn:
36     cur.execute(f"""CREATE TABLE IF NOT EXISTS users_admin (
37         id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE NOT NULL,
38         user_id INTEGER NOT NULL,
39         state INTEGER,
40         data TEXT
41     )""")

```

Рисунок 3.28 – Код таблиці users_admin

Код створення таблиці news на рис. 3.29. Таблиця має такі поля:

- id - поле зберігає в собі id новини;
- text_news – текст новини;
- id_photo - зберігає id зображення у телеграмм.

```

db_file = "db.db"
conn = sqlite3.connect(db_file)
cur = conn.cursor()
with conn:
    cur.execute(f"""CREATE TABLE IF NOT EXISTS news (
                        id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE NOT NULL,
                        text_news TEXT,
                        id_photo TEXT
                    )""")

```

Рисунок 3.29 – Код таблиці news

Таблиця reviews має такі поля:

- id - поле зберігає в собі id відгуку;
- text_users – текст відгуку;
- id_photo - зберігає id зображення у телеграм;
- moder – поле мацерації, для публікації.

На рисунку 3.30 зображено код створення таблиці.

```

db_file = "db.db"
conn = sqlite3.connect(db_file)
cur = conn.cursor()
with conn:
    cur.execute(f"""CREATE TABLE IF NOT EXISTS reviews (
                        id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE NOT NULL,
                        text_users TEXT,
                        id_photo TEXT,
                        moder INTEGER
                    )""")

```

Рисунок 3.30 – Код таблиці reviews

Таблиця work має такі поля:

- id - поле зберігає в собі id вакансії;
- text_work – текст вакансії;
- moder – поле мацерації, для публікації;

На рисунку 3.31 зображено код створення таблиці.

```

db_file = "db.db"
conn = sqlite3.connect(db_file)
cur = conn.cursor()
with conn:
    cur.execute(f"""CREATE TABLE IF NOT EXISTS work (
                                id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE NOT NULL,
                                text_work TEXT,
                                moder INTEGER
                            )""")

```

Рисунок 3.31 – Код таблиці work

У функції `check_user_id(user_id)` я перевіряю наявність даних про користувача в базі даних. Якщо його не, то викликається `add_user(user_id)`, куди передаю отриманий `user_id` і додаю користувача в базу.

Функція `get_state_users(_user_id)` повертає мені статус користувача, що біот міг визначити, що він має зараз робити і які відповіді мають бути рис. 3.32.

Далі йдуть функції для роботи з тимчасовими даними `get_data(_user_id)` - отримує ID користувача і повертає його data, а `update_user_data(_user_id,_data,_state)` - отримує ID, змінний data, змінений статус і перезаписує в базі даних рис. 3.33.

Як говорилося раніше, до меню адміністратора мають доступ, тільки ті, хто є в таблиці `user_admin`, тому коли викликається команда `"/admin"` потрібно перевірити чи є користувач адміністратором.

Функція `check_admin_user_id(user_id)` - якраз і перевіряє та повертає відповідь, чи є користувач адміністратором рис. 3.34.

```
def get_state_users(_user_id):
    db_file = "db.db"
    conn = None
    data = None
    try:
        sql = f"""SELECT * FROM users WHERE user_id = {_user_id}"""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchone()[1]
        cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()
        return data
```

Рисунок 3.32 – Код отримання стану користувача

```
def get_data(_user_id):
    db_file = "db.db"
    conn = None
    data = None
    try:
        sql = f"""SELECT * FROM users WHERE user_id = {_user_id}"""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchone()[2]
        cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()
        print(data)
        return data

def update_user_data(_user_id, _data, _state):
    db_file = "db.db"
    conn = None
    try:
        sql = f"""UPDATE users SET data = "{_data}", state = "{_state}" WHERE user_id = {_user_id}"""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        conn.commit()
        cur.close()
    except Exception as e:
        print(e, 6)
    finally:
        if conn is not None:
            conn.close()
```

Рисунок 3.33 – Код data users

```

def check_admin_user_id(user_id):
    db_file = "db.db"
    conn = None
    check = False
    try:
        sql = f"SELECT * FROM users_admin WHERE user_id = {user_id}"
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchone()
        if data != None:
            cur.close()
            check = True
        else:
            cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()
    return check

```

Рисунок 3.34– Код adminmenu

Функція `get_state_users_admin(user_id)` - отримуємо статус, щоб визначити етап, на якому працює адміністратор.

Функція `up_state_users_admin(_state, _user_id)` – оновлює статус адміністратора в базі даних.

Функція `get_product_inline(category)` – видає інформацію про товари, що відповідають отриманій категорії товарів.

Функція `get_add_ad(_user_id)` - тут ми передаємо id користувача, який додає товар. Після чого ми в цій функції викликаємо `get_data()`, яка поверне нам дані, які користувач вводив і на їх основі створює новий товар у таблиці `products`.

ВИСНОВКИ

У ході експлуатації програмного продукту критичних помилок не було виявлено.

Бот дозволяє знайти всю цікаву інформацію для користувача в сфері кальянів, але планується подальший розвиток продукту. Буду покращуватися виведення відгуків і вакансій з подальшим додаванням фільтра. Також буде покращено фільтрацію оголошень для більш комфортного пошуку.

Буде доданий ще блок порад, де люди зможуть залишати свої поради щодо тематики кальянів. Планується зі зростанням аудиторії, додаванням рекламних постів та можливості просування своїх оголошень.

На даному етапі є унікальний продукт, аналога, якому немає. Реалізований інтерфейс легко та пропонує найнеобхідніше для користувача.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Всё, о чём должен знать разработчик Телеграм-ботов [Электрон. ресурс]. – Режим доступа: <https://habr.com/ru/post/543676/>.
2. Telegram [Электрон. ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Telegram>.
3. Феномен Телеграм-ботов: что это, зачем нужны, примеры [Электрон. ресурс]. – Режим доступа: <https://1ps.ru/blog/dirs/2021/fenomen-telegram-botov-chto-eto-zachem-nuzhnyi-primeryi/>.
4. Мова програмування Python [Електрон. ресурс]. – Режим доступа: <https://uk.wikipedia.org/wiki/Python>.
5. Итоги курса "Объектно-ориентированное программирование на Python" [Электрон. ресурс]. – Режим доступа: <https://younglinux.info/oopython/oop2>.
6. Преимущества Python [Электрон. ресурс]. – Режим доступа: <https://www.digest.pro/news/preimushhestva-python-pered-drugimi-jazykami-programmirovaniya/#:~:text=Кроссплатформенный%3A%20подходит%20для%20разных%20платформ,обучения%2C%20в%20области%20интернета%20вещей>.
7. Каковы недостатки Python? [Электрон. ресурс]. – Режим доступа: <https://pythobyte.com/what-are-pythons-disadvantages-31f03aa4/>.
8. Инкапсуляция, атрибуты и свойства [Электрон. ресурс]. – Режим доступа: <https://metanit.com/python/tutorial/7.2.php>.
9. Лучшие IDE и редакторы кода для Python [Электрон. ресурс]. – Режим доступа: <https://tproger.ru/translations/python-ide/>.
10. Выбираем базу данных [Электрон. ресурс]. – Режим доступа: <https://medium.com/nuances-of-programming/как-выбрать-правильную-базу-данных-62b0747f7f47>.

11. Плюсы и минусы MongoDB? – Режим доступа: <https://overcoder.net/q/8376/плюсы-и-минусы-mongodb>.
12. Что такое PostgreSQL? [Электрон. ресурс]. – Режим доступа: <https://ru.education-wiki.com/5154595-what-is-postgresql>.
13. SQLite [Электрон. ресурс]. – Режим доступа: <https://blog.skillfactory.ru/glossary/sqlite/>.
14. PEP 8 Python [Electronic source]. – Access mode: <https://realpython.com/python-pep8/#:~:text=PEP%208%2C%20sometimes%20spelled%20PEP8,and%20consistency%20of%20Python%20code>.
15. Синтаксис языка Python [Электрон. ресурс]. – Режим доступа: <https://pythonworld.ru/osnovy/sintaksis-yazyka-python.html>.
16. Платіжна система Telegram [Электрон. ресурс]. – Режим доступа: https://github.com/icool5520/Yoga_bot/blob/master/magaz1.py.
17. InlineMode Telegram и реализация InlineQueryResultArticle [Электрон. ресурс]. – Режим доступа: https://github.com/icool5520/Yoga_bot/blob/master/yoga.py.

ДОДАТОК А
Текст програми

A.1 Компонент main.py

```

import telebot
from telebot import types
from uuid import uuid4
from telegram import InlineQueryResultPhoto [17]
import base64
import requests
import time
import ast
import json
from telebot.types import LabeledPrice, ShippingOption [16]
import markup
import db_cmd

bot = telebot.TeleBot('1581169892:AAGb2y62DodrYfOzHmEYfRIxTjn2onTpwZ0')

imgBBKey = '3a44091aef2c77f10f9cb6734dd30e88'
provider_token = '410694247:TEST:6a469bf3-cb75-49ea-b000-c89bab19fb71'

#-----COMMANDS-----
-----
@bot.message_handler(commands=['admin'])
def message_handler(message):
    cid = message.chat.id
    data = db_cmd.check_admin_user_id(cid)
    if data == True:
        bot.send_message(message.chat.id, "Админ панель",
                           reply_markup=markup.gen_adminmenu())
    else:
        bot.send_message(message.chat.id, text="Нет доступа")

@bot.message_handler(commands=['delet'])
def message_handler(message):

    cid = message.chat.id
    mid = message.message_id
    uid = message.from_user.id
    stopmid = db_cmd.get_mid(uid)
    for i in range(mid, stopmid, -1):

```

```

try:
    print(i)
    bot.delete_message(cid, i)
    db_cmd.update_user_mid(uid, mid)
except Exception as e:
    mid = mid - 1
    print('delet:', e)
    bot.send_message(message.chat.id, "Приветствуем в HookahHome.\nЗдесь мы
собрали все что может быть вам интересно по кальяной индустрии.\nДля
обратной связи @tedillumDevTelegam.",
        reply_markup=markup.gen_mainmenu())

@bot.message_handler(commands=['start'])
def message_handler(message):
    cid = message.chat.id
    db_cmd.check_user_id(cid)
    bot.send_message(message.chat.id, "Приветствуем в HookahHome.\n"+
        "Здесь мы собрали все что может быть вам интересно по кальяной
индустрии.\nДля обратной связи @tedillumDevTelegam.",
        reply_markup=markup.gen_mainmenu())

@bot.callback_query_handler(func=lambda call: call.data == "start")
def callback_mainmenu(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            bot.edit_message_text(
                chat_id=cid,
                message_id=mid,
                text="Приветствуем в HookahHome.\nЗдесь мы собрали все что может
быть вам интересно по кальяной индустрии.\nДля обратной связи
@tedillumDevTelegam.",
                parse_mode="Markdown",
                reply_markup=markup.gen_mainmenu())
    except Exception as e:
        print('callback_mainmenu:', e)
#-----BUY-----
-----

```

```
@bot.callback_query_handler(func=lambda call: call.data == "buy")
def callback_buy(call):
```

```
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            bot.edit_message_text(
                chat_id=cid,
                message_id=mid,
                text=" Выберите товар ",
                parse_mode="Markdown",
                reply_markup=markup.categor())
    except Exception as e:
        print('callback_buy:', e)
```

```
@bot.inline_handler(lambda query: types.InlineQuery) [17]
def query_text(inline_query):
```

```
    try:
        if inline_query.query=="start":
            uid = inline_query.from_user.id
            db_cmd.check_user_id(uid)
            bot.send_message(
                chat_id=uid,
                text="Приветствуем в HookahHome.\nЗдесь мы собрали все что может
                быть вам интересно по кальяной индустрии.\nДля обратной связи
                @tedillumDevTelegam.",
                parse_mode="Markdown",
                reply_markup=markup.gen_mainmenu())
        elif inline_query.query=="news":
            data = db_cmd.get_news()
            list_inline=[]
            print(data)
            for i in data:
                text = f'{i[1]}'
                r = types.InlineQueryResultArticle(
                    id=str(uuid4()),
                    title =str(i[1]),
                    description=f"Описание",
                    thumb_url=f"{i[2]}",
```

```

input_message_content=types.InputTextMessageContent(message_text=f"{text}\n'["
({i[2]}))", parse_mode='Markdown'),
        reply_markup=markup.categor())
        list_inline.append(r)
        bot.answer_inline_query(inline_query.id,
list_inline,0,switch_pm_parameter="start")
    else:
        data = db_cmd.get_product_inline(inline_query.query.split('/')[0])
        list_inline=[]
        for i in data:
            if inline_query.query.split('/')[1]==" :
                text = f'Название: {i[3]}\nЦена: {i[4]}\nКомплектация:
{i[5]}\nКонтакты: {i[6]}'
                r = types.InlineQueryResultArticle(
                    id=str(uuid4()),
                    title =str(i[3]),
                    description=f'Цена: {i[4]}",
                    thumb_url=f"{i[7]}",

input_message_content=types.InputTextMessageContent(message_text=f"{text}\n'["
({i[7]}))", parse_mode='Markdown'),
        reply_markup=markup.categor())
        print(text)
        list_inline.append(r)
        video = i[2]
        elif (inline_query.query.split('/')[1]).isdigit()==True and
i[4]>=int(inline_query.query.split('/')[1]):
            text = f'Название: {i[3]}\nЦена: {i[4]}\nКомплектация:
{i[5]}\nКонтакты: {i[6]}'
            r = types.InlineQueryResultArticle(
                id=str(uuid4()),
                title =str(i[3]),
                description=f'Цена: {i[4]}",
                thumb_url=f"{i[7]}",

input_message_content=types.InputTextMessageContent(message_text=f"{text}\n'["
({i[7]}))", parse_mode='Markdown'),
        reply_markup=markup.categor())
        print(text)
        list_inline.append(r)

```

```

        elif (inline_query.query.split('/')[1]).isdigit()==False and
i[3].count(inline_query.query.split('/')[1])>0:
            text = f'Название: {i[3]}\nЦена: {i[4]}\nКомплектация:
{i[5]}\nКонтакты: {i[6]}'
            r = types.InlineQueryResultArticle(
                id=str(uuid4()),
                title =str(i[3]),
                description=f'Цена: {i[4]}',
                thumb_url=f'{i[7]}',

input_message_content=types.InputTextMessageContent(message_text=f'{text}\n'[:-1]
({i[7]}))", parse_mode='Markdown'),
            reply_markup=markup.categor())
            print(text)
            list_inline.append(r)
            bot.answer_inline_query(inline_query.id,
list_inline,0,switch_pm_parameter="start")
            bot.send_video(video)
        except Exception as e:
            print(e)
#-----SALE-----
-----
@bot.callback_query_handler(func=lambda call: call.data == "sall")
def callback_shop(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            seach_info = str(db_cmd.get_data(uid))
            db_cmd.update_user_data(uid, seach_info, 'sall')
            bot.edit_message_text(
                chat_id=cid,
                message_id=mid,
                text=" Укажите категорию товара ",
                parse_mode="Markdown",
                reply_markup=markup.categor_sale())
    except Exception as e:
        print('callback_shop:', e)

```

```

@bot.callback_query_handler(func=lambda call: call.data == "hookah" or call.data == "bowls" or call.data == "tobac" or call.data == "colbs" or call.data == "other")
def callback_categor(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            categor = str(call.data)
            data = db_cmd.get_data(uid)
            d2 = dict(ast.literal_eval(data))
            d2["categor"] = categor
            state = db_cmd.get_state_users(uid)
            if state == 'sall':
                db_cmd.update_user_data(str(uid), str(d2), 'add_brand')
                bot.send_message(cid, "Введите бренд товара ",
                                reply_markup=markup.set_news())
    except Exception as e:
        print('callback_categor:', e)

```

```

@bot.callback_query_handler(func=lambda call: call.data == "yes" or call.data == "no")
def callback_gomoder(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            if call.data == "no":
                bot.edit_message_text(message.chat.id, "Приветствуем в
HookahHome.\nЗдесь мы собрали все что может быть вам интересно по
кальяной индустрии.\nДля обратной связи @tedillumDevTelegam.",
                                reply_markup=markup.gen_mainmenu())
            elif call.data == "yes":
                state_user = db_cmd.get_state_users(uid)
                if state_user == 'add_prod':
                    db_cmd.get_add_ad(uid)
                    d2=db_cmd.get_data(uid)

```

```

        bot.send_message(cid, "Ваше объявление отправленно на
модерацию!")
        time.sleep(3)
        stopmid = db_cmd.get_mid(uid)
        db_cmd.update_user_data(str(uid), str(d2),0)
        for i in range(mid, stopmid, -1):
            try:
                print(i)
                bot.delete_message(cid, i)
                db_cmd.update_user_mid(uid, mid)
            except Exception as e:
                mid = mid - 1
                print('delet:', e)
        bot.send_message(cid, "Приветствуем в HookahHome.\nЗдесь мы
собрали все что может быть вам интересно по кальяной индустрии.\nДля
обратной связи @tedillumDevTelegam.",
            reply_markup=markup.gen_mainmenu())
    except Exception as e:
        print('callback_gomoder:', e)

```

```

@bot.callback_query_handler(func=lambda call: call.data == "news")
def callback_news(call):

```

```

    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)

            bot.answer_inline_query(inline_query.id,
list_inline,0,switch_pm_parameter="start")
            except Exception as e:
                print('callback_news:', e)

```

```

#-----ADMINMENU-----

```

```

@bot.callback_query_handler(func=lambda call: call.data == "adminnews")
def callback_adminnews(call):

```

```

    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id

```

```

if uid == cid:
    db_cmd.check_user_id(cid)
    data = db_cmd.get_state_users_admin(uid)
    db_cmd.up_state_users_admin('add_text_news', uid)
    bot.edit_message_text(
        chat_id=cid,
        message_id=mid,
        text="Введите текст новости",
        parse_mode="Markdown")
except Exception as e:
    print('callback_adminnews:', e)

@bot.callback_query_handler(func=lambda call: call.data == "moder_sall")
def callback_moder_sall(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            db_cmd.up_state_users_admin('add_prod', uid)
            data = db_cmd.get_moder_product()
            for i in data:
                photo = i[7]
                bot.send_photo(uid, photo)
                bot.send_message(
                    cid, text=f"ID: {i[0]}\n Название: {i[3]}\n Цена: {i[4]}\n Описание:
{i[5]}\n Контакты: {i[6]}")
                bot.send_message(cid, "Введите через запятую id которые опубликовать
или слово (all) если все")
            except Exception as e:
                print('callback_adminnews:', e)

@bot.callback_query_handler(func=lambda call: call.data == "adminreviews")
def callback_adminreviews(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            db_cmd.up_state_users_admin('moder_reviews', uid)

```

```

data = db_cmd.get_reviews_admin()
for i in data:
    photo = i[2]
    bot.send_photo(uid, photo)
    bot.send_message(
        cid, text=f"ID: {i[0]}\n Текст: {i[1]}\n")
    bot.send_message(cid, "Введите через запятую id которые публиковать
или слово (all) если все")

```

```

except Exception as e:
    print('callback_adminreviews:', e)

```

#-----WORK-----

```

@bot.callback_query_handler(func=lambda call: call.data == "work")
def callback_news(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            bot.send_message(
                cid, text=f"Вакансии: ",reply_markup=markup.gen_work())
    except Exception as e:
        print('callback_news:', e)
@bot.callback_query_handler(func=lambda call: call.data == "seach_work")
def callback_news(call):

```

```

    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            data = db_cmd.get_work_moder()
            for i in data:
                bot.send_message(
                    cid, text=f"{i}",reply_markup=markup.set_news())
    except Exception as e:
        print('callback_news:', e)
@bot.callback_query_handler(func=lambda call: call.data == "add_work")
def callback_news(call):
    try:

```

```

cid = call.message.chat.id
uid = call.from_user.id
mid = call.message.message_id
if uid == cid:
    db_cmd.check_user_id(cid)
    data = db_cmd.get_data(uid)
    d2 = dict(ast.literal_eval(data))
    d2["brand"] = ""
    db_cmd.update_user_data(str(uid), str(d2), 'add_text_work')
    bot.send_message(
        cid, text=f"Введите информацию о вакансии: ")
except Exception as e:
    print('callback_news:', e)
@bot.callback_query_handler(func=lambda call: call.data == "moder_work")
def callback_moder_work(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            db_cmd.up_state_users_admin('moder_work',uid)
            data = db_cmd.get_work_admin()
            print(data)
            for i in data:
                bot.send_message(
                    cid, text=f"ID: {i[0]}\n Текст: {i[1]}\n")
                bot.send_message(cid, "Введите через запятую id которые публиковать
или слово (all) если все")

    except Exception as e:
        print('callback_moder_work:', e)
#-----REVIEWS-----
@bot.callback_query_handler(func=lambda call: call.data == "reviews")
def callback_reviews(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)

```

```

        bot.send_message(cid, "Выберите
действие", reply_markup=markup.get_reviews())
    except Exception as e:
        print('callback_reviews:', e)

@bot.callback_query_handler(func=lambda call: call.data == "add_reviews")
def callback_add_reviews(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            data = db_cmd.get_data(uid)
            d2 = dict(ast.literal_eval(data))
            d2["brand"] = ""
            db_cmd.update_user_data(str(uid), str(d2), 'add_text_reviews')
            bot.send_message(cid, "Введите текст")
    except Exception as e:
        print('callback_add_reviews:', e)

@bot.callback_query_handler(func=lambda call: call.data == "prt_reviews")
def callback_prt_reviews(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        if uid == cid:
            db_cmd.check_user_id(cid)
            data = db_cmd.get_reviews_moder()
            for i in data:
                photo = i[2]
                bot.send_photo(uid, photo)
                bot.send_message(
                    cid, text=f"{i[1]}")
    except Exception as e:
        print('callback_prt_reviews:', e)
#-----MESSAGE-----

@bot.message_handler(func=lambda m: True)
def echo_all(message):
    try:
        cid = message.chat.id

```

```

uid = message.from_user.id
mid = message.message_id
if uid == cid:
    db_cmd.check_user_id(cid)
    state = data = db_cmd.get_state_users_admin(uid)
    state_user = db_cmd.get_state_users(uid)
    if state == 'add_text_news':
        text_news = message.text
        db_cmd.up_text_users_admin(text_news, uid)
        db_cmd.up_state_users_admin('add_img_new', uid)
        bot.send_message(
            chat_id=cid,
            text=" Добавить изображение ",
            parse_mode="Markdown",
            reply_markup=markup.set_img())
    elif state == 'moder_reviews':
        id_reviews = (message.text).split(',')
        for i in id_reviews:
            db_cmd.up_id_moder_reviews(i)
            db_cmd.up_state_users_admin(0, uid)
            bot.send_message(
                chat_id=cid,
                text=" Опубликовано ",
                parse_mode="Markdown",
                reply_markup=markup.gen_adminmenu())
    elif state == 'moder_work':
        id_work = (message.text).split(',')
        for i in id_work:
            db_cmd.up_id_moder_work(i)
            db_cmd.up_state_users_admin(0, uid)
            bot.send_message(
                chat_id=cid,
                text=" Опубликовано ",
                parse_mode="Markdown",
                reply_markup=markup.gen_adminmenu())
    elif state_user == 'add_text_work':
        text_work = message.text
        data = db_cmd.get_data(uid)
        d2 = dict(ast.literal_eval(data))
        d2["brand"] = ""
        db_cmd.update_user_data(str(uid), str(d2), '0')
        db_cmd.set_works(text_work)

```

```

bot.send_message(
    chat_id=cid,
    text=" Вакансии отправленна на модерацию ",
    parse_mode="Markdown",
    reply_markup=markup.gen_mainmenu())

elif state_user == 'add_text_reviews':
    text_review = message.text
    data = db_cmd.get_data(uid)
    d2 = dict(ast.literal_eval(data))
    d2["brand"] = ""
    db_cmd.update_user_data(str(uid), str(d2), 'add_img_reviews')
    db_cmd.get_text_users(text_review, uid)
    bot.send_message(
        chat_id=cid,
        text=" Добавить изображение ",
        parse_mode="Markdown",
        reply_markup=markup.set_img())
elif state == 'add_prod':
    text_news = message.text
    if text_news == 'all':
        db_cmd.up_all_moder_admin()
        id_prod = (message.text).split(',')
    else:
        for i in id_prod:
            db_cmd.up_id_moder_prod(i)
            db_cmd.up_state_users_admin(0, uid)
            bot.send_message(
                chat_id=cid,
                text=" Опубликованно ",
                parse_mode="Markdown",
                reply_markup=markup.gen_adminmenu())
if state_user == 'add_brand':
    data = db_cmd.get_data(uid)
    d2 = dict(ast.literal_eval(data))
    print(message.text)
    d2["brand"] = message.text
    print(d2)
    db_cmd.update_user_data(str(uid), str(d2), 'add_price')
    bot.send_message(
        cid,
        text=" Введите цену ващего товара ",

```

```

        parse_mode="Markdown",
        reply_markup=markup.set_news())
elif state_user == 'add_price':
    data = db_cmd.get_data(uid)
    d2 = dict(ast.literal_eval(data))
    if (message.text).isdigit() == True:
        d2["price"] = message.text
        db_cmd.update_user_data(str(uid), str(d2), 'add_dicr')
        bot.send_message(
            cid,
            text=" Введите опиисание о товаре(Высота,материал,состояние и
тд) ",
            parse_mode="Markdown",
            reply_markup=markup.set_news())
else:
    bot.send_message(
        cid,
        text=" Введите ценну числами! ",
        parse_mode="Markdown",
        reply_markup=markup.set_news())
elif state_user == 'add_dicr':
    data = db_cmd.get_data(uid)
    d2 = dict(ast.literal_eval(data))
    d2["discrip"] = message.text
    db_cmd.update_user_data(str(uid), str(d2), 'add_contact')
    bot.send_message(
        cid,
        text=" Оставьте свои контакты(Для удобства советуем оставить ваш
user name в Теллеграмм)",
        parse_mode="Markdown",
        reply_markup=markup.set_news())
elif state_user == 'add_contact':
    data = db_cmd.get_data(uid)
    d2 = dict(ast.literal_eval(data))
    d2["contact"] = message.text
    state = db_cmd.get_state_users(uid)
    db_cmd.update_user_data(str(uid), str(d2), 'add_img_prod')
    bot.send_message(
        chat_id=cid,
        text=" Добавить изображение ",
        parse_mode="Markdown",
        reply_markup=markup.set_img())

```

```

elif state_user == 'pay_for_prog':
    if message.text.isdigit() == True:
        prices = [LabeledPrice(label='Оплата товара', amount=int(message.text)
* 100), LabeledPrice('Доставка', 0)]
        bot.send_invoice(cid, title='Поддержка проекта',
            description='Подтверждение Оплаты',
            provider_token=provider_token,
            currency='uah',
            photo_url='https://usehelp.org/wp-
content/uploads/2019/04/%D1%81%D0%BF%D0%B0%D1%81%D0%B8%D0%B1
%D0%BE-%D1%87%D1%82%D0%BE-
%D0%BF%D0%BE%D0%BC%D0%BE%D0%B3%D0%B0%D0%B5%D1%82%D
0%B5-1.png',
            photo_height=512, # !=0/None or picture won't be shown
            photo_width=512,
            photo_size=512,
            is_flexible=False, # True If you need to set up Shipping Fee
            prices=prices,
            start_parameter='donat',
            invoice_payload='HAPPY FRIDAYS COUPON')
    else:
        bot.send_message(cid, "Введите сумму доната цифрами")
except Exception as e:
    print('callback_add:', e)

```

```

@bot.message_handler(content_types=['photo'])
def callback_adimg(message):
    try:
        cid = message.chat.id
        uid = message.from_user.id
        mid = message.message_id
        if uid == cid:
            state = db_cmd.get_state_users_admin(uid)
            state_user = db_cmd.get_state_users(uid)
            db_cmd.check_user_id(cid)
            if state == 'add_img_new':
                file_info = bot.get_file(
                    message.photo[len(message.photo) - 1].file_id)
                img = file_info.file_id
                text_news = db_cmd.get_text_admin(uid)
                db_cmd.set_news(text_news, img)

```

```

db_cmd.up_state_users_admin(0, uid)
bot.send_message(
    chat_id=cid,
    text=" Новочть добавлена ",
    parse_mode="Markdown",
    reply_markup=markup.gen_adminmenu())
if state_user == 'add_img_reviews':
    file_info = bot.get_file(
        message.photo[len(message.photo) - 1].file_id)
    img = file_info.file_id
    text_news = db_cmd.get_reviews_user(uid)
    db_cmd.set_reviews(text_news, img)
    db_cmd.up_state_users_admin(0, uid)
    bot.send_message(
        chat_id=cid,
        text=" Отзыв добавлен ",
        parse_mode="Markdown",
        reply_markup=markup.gen_mainmenu())
if state_user == 'add_img_prod':
    file_info = bot.get_file(
        message.photo[len(message.photo) - 1].file_id)
    img = bot.download_file(file_info.file_path)
    src=file_info.file_path;
    with open(src, 'wb') as new_file:
        new_file.write(img)

    with open(f"photos/{(file_info.file_path).split('/')[1]}", "rb") as file:
        url = "https://api.imgbb.com/1/upload"
        payload = {
            "key": imgBBKey,
            "image": base64.b64encode(file.read()),
        }
        res = requests.post(url, payload)
        resp = res.json()
    data = db_cmd.get_data(uid)
    d2 = dict(ast.literal_eval(data))
    d2["photo"] = resp["data"]["url"]
    db_cmd.update_user_data(str(uid), str(d2), 'add_prod')
    bot.send_message(
        chat_id=cid,
        text=" Отправить на модерацию? ",
        parse_mode="Markdown",

```

```

        reply_markup=markup.get_add())
except Exception as e:
    print('callback_prt_reviews:', e)

#-----PAY-----
----- [16]
@bot.callback_query_handler(func=lambda call: call.data == "help")
def callback_pay(call):
    try:
        cid = call.message.chat.id
        uid = call.from_user.id
        mid = call.message.message_id
        db_cmd.update_user_data(uid,"{'categor': 'tobac', 'brand': '', 'price': '', 'discrip': '',
'contact': '', 'photo': ''}','pay_for_prog')
        bot.send_message(cid,"Введите сумму для пожертвований: ")
    except Exception as e:
        print('callback_pay:', e)

@bot.shipping_query_handler(func=lambda query: True)
def shipping(shipping_query):

    cid = shipping_query.message.chat.id
    uid = shipping_query.from_user.id
    mid = shipping_query.message.message_id
    print(shipping_query)
    bot.answer_shipping_query(shipping_query.id, ok=True,
shipping_options=shipping_options,
                           error_message='О, кажется, наши курьеры Собаки сейчас
обедают. Попробуйте позже!')

@bot.pre_checkout_query_handler(func=lambda query: True)
def checkout(pre_checkout_query):
    bot.answer_pre_checkout_query(pre_checkout_query.id, ok=True,
                           error_message="Инопланетяне пытались украсть CVV вашей
карты, но мы успешно защитили ваши учетные данные")

@bot.message_handler(content_types=['successful_payment'])
def got_payment(message):
    cid = message.chat.id

```

```

uid = message.from_user.id
mid = message.message_id
bot.send_message(message.chat.id,
                  'Hoooooray! Thanks for payment! We will proceed your order for `{}`
as fast as possible! '.format(
    message.successful_payment.total_amount / 100,
    message.successful_payment.currency),
    parse_mode='Markdown')

bot.send_message(
    chat_id=cid,
    text="Приветствуем в HookahHome.\nЗдесь мы собрали все что может
быть вам интересно по кальянной индустрии.\nДля обратной связи
@tedillumDevTelegam.",
    parse_mode="Markdown",
    reply_markup=markup.gen_mainmenu())

bot.skip_pending = True

bot.polling()

```

A.2 Компонент db_cmd.py

```

import sqlite3
import ast

# Создание базы данных если ее нет

def create_tables():
    db_file = "db.db"
    conn = sqlite3.connect(db_file)
    cur = conn.cursor()
    with conn:
        cur.execute(f"""CREATE TABLE IF NOT EXISTS users (
            user_id INTEGER PRIMARY KEY NOT NULL,
            state INTEGER NOT NULL,

```



```

        state INTEGER,
        data TEXT
    )""")
db_file = "db.db"
conn = sqlite3.connect(db_file)
cur = conn.cursor()
with conn:
    cur.execute(f"""\CREATE TABLE IF NOT EXISTS news (
                                id INTEGER PRIMARY KEY
AUTOINCREMENT UNIQUE NOT NULL,
                                text_news TEXT,
                                id_photo TEXT
                                )""")
db_file = "db.db"
conn = sqlite3.connect(db_file)
cur = conn.cursor()
with conn:
    cur.execute(f"""\CREATE TABLE IF NOT EXISTS reviews (
                                id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE
NOT NULL,
                                text_users TEXT,
                                id_photo TEXT,
                                moder INTEGER
                                )""")
db_file = "db.db"
conn = sqlite3.connect(db_file)
cur = conn.cursor()
with conn:

```

```

cur.execute(f"""CREATE TABLE IF NOT EXISTS work (
                id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE
NOT NULL,
                text_work TEXT,
                moder INTEGER
            )""")

```

```

#-----USER-----

```

```

def check_user_id(user_id):
    db_file = "db.db"
    conn = None
    check = False
    try:
        sql = f"""SELECT * FROM users WHERE user_id = {user_id}"""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchone()
        if not data:
            cur.close()
            add_user(user_id)
            check = True
        else:
            cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()

```

return check

```
def add_user(user_id):
    db_file = "db.db"
    conn = None
    try:
        sql = """INSERT INTO users(user_id, state, data,mid) VALUES(?, ?, ?,
?);"""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        mass = {'categor': ", 'brand': ", 'price': ", 'discrip': ", 'contact': ", 'photo': "}
        cur.execute(sql, (user_id, 0,str(mass),0))
        conn.commit()
        cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()

def get_state_users(_user_id):
    db_file = "db.db"
    conn = None
    data = None
    try:
        sql = f"""SELECT * FROM users WHERE user_id = {_user_id}"""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
```

```

        cur.execute(sql)
        data = cur.fetchone()[1]
        cur.close()
except Exception as e:
    print(e, 1)
finally:
    if conn is not None:
        conn.close()
    return data

def get_data(_user_id):
    db_file = "db.db"
    conn = None
    data = None
    try:
        sql = f""""SELECT * FROM users WHERE user_id = {_user_id}""""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchone()[2]
        cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()
        print(data)
        return data

```

```

def update_user_data(_user_id,_data,_state):
    db_file = "db.db"
    conn = None
    try:
        sql = f""""UPDATE users SET data = "{_data}", state = "{_state}" WHERE
user_id = {_user_id}"""""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        conn.commit()
        cur.close()
    except Exception as e:
        print(e, 6)
    finally:
        if conn is not None:
            conn.close()

#-----ADMINMENU-----

def check_admin_user_id(user_id):
    db_file = "db.db"
    conn = None
    check = False
    try:
        sql = f""""SELECT * FROM users_admin WHERE user_id = {user_id}"""""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchone()

```

```

        if data != None:
            cur.close()
            check = True
        else:
            cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()
        return check

def get_state_users_admin(_user_id):
    db_file = "db.db"
    conn = None
    data = None
    try:
        sql = f"""SELECT state FROM users_admin WHERE
user_id='{_user_id}'"""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchall()
        print(data)
        cur.close()
    except Exception as e:
        print(e, 1)
    return 0

```

```

finally:
    if conn is not None:
        conn.close()
    if len(data)<1:
        return 0
    else:
        return data[0][0]

def up_state_users_admin(_state,_user_id):
    db_file = "db.db"
    conn = None
    try:
        sql = f""""UPDATE users_admin SET state = "{_state}" WHERE user_id =
{_user_id}""""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        conn.commit()
        cur.close()
    except Exception as e:
        print(e, 6)
    finally:
        if conn is not None:
            conn.close()

def up_text_users_admin(_text_news,_user_id):
    db_file = "db.db"
    conn = None
    try:

```

```

        sql = f""""UPDATE users_admin SET data = "{_text_news}" WHERE
user_id = {_user_id}""""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        conn.commit()
        cur.close()
    except Exception as e:
        print(e, 6)
    finally:
        if conn is not None:
            conn.close()
def get_text_admin(_user_id):
    db_file = "db.db"
    conn = None
    data = None
    try:
        sql = f""""SELECT data FROM users_admin WHERE user_id='{_user_id}""""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchall()
        cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()

```

```

        return data[0][0]

def get_moder_product():
    db_file = "db.db"
    conn = None
    data = None
    try:
        sql = f""""SELECT * FROM product WHERE moder=0""""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchall()
        cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()
    return data

def up_all_moder_admin():
    db_file = "db.db"
    conn = None
    try:
        sql = f""""UPDATE product SET moder = 1 WHERE moder = 0""""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        conn.commit()

```

```

        cur.close()
    except Exception as e:
        print(e, 6)
    finally:
        if conn is not None:
            conn.close()
def up_id_moder_prod(id_prod):
    db_file = "db.db"
    conn = None
    try:
        sql = f"UPDATE product SET moder = 1 WHERE id = \"{id_prod}\""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        conn.commit()
        cur.close()
    except Exception as e:
        print(e, 6)
    finally:
        if conn is not None:
            conn.close()

#-----PRODUCT-----
def get_product_inline(_categor):
    db_file = "db.db"
    conn = None
    data = None
    try:

```

```

        sql = f""""SELECT * FROM product WHERE categor = '{_categor}' AND
moder=1""""

        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchall()
        cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()
        return data

def get_add_ad(_user_id):
    db_file = "db.db"
    conn = None
    data = get_data(_user_id)
    datax = dict(ast.literal_eval(data))
    print(datax)
    categor = datax["categor"]
    brand = datax["brand"]
    discrip = datax["discrip"]
    contact = datax["contact"]
    price = datax["price"]
    photo = datax["photo"]
    try:

```

```

    sql = """INSERT INTO product(user_id , categor , brand , price, discrip ,
contact, photo, moder) VALUES(?, ?, ?,?,?,?,?);"""
    conn = sqlite3.connect(db_file)
    cur = conn.cursor()

    cur.execute(sql, (_user_id, categor, brand, price,
                      discrip, contact, photo, False))

    conn.commit()
    cur.close()
except Exception as e:
    print(e, 2)
finally:
    if conn is not None:
        conn.close()

#-----DELET-----
def get_mid(_user_id):
    db_file = "db.db"
    conn = None
    data = None
    try:
        sql = f"""SELECT * FROM users WHERE user_id = {_user_id}"""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchone()[4]
        cur.close()
        print(data)
    except Exception as e:

```

```

        print(e, 1)
    finally:
        if conn is not None:
            conn.close()
        return data
def update_user_mid(_user_id,_mid):
    db_file = "db.db"
    conn = None
    try:
        sql = f""""UPDATE users SET mid = '{_mid}' WHERE user_id =
'{_user_id}'""""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        conn.commit()
        cur.close()
    except Exception as e:
        print(e, 6)
    finally:
        if conn is not None:
            conn.close()

#-----REVIEWS-----
def set_reviews(_text,_photo):
    db_file = "db.db"
    conn = None
    try:
        sql = """"INSERT INTO reviews(text_users, id_photo, moder)
VALUES(?,?,?);""""

```

```

    conn = sqlite3.connect(db_file)
    cur = conn.cursor()
    cur.execute(sql, (_text,_photo,0))
    conn.commit()
    cur.close()
except Exception as e:
    print(e, 1)
finally:
    if conn is not None:
        conn.close()

def get_reviews_moder():
    db_file = "db.db"
    conn = None
    data = None
    try:
        sql = f""""SELECT * FROM reviews WHERE moder = 1""""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchall()
        cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()
    return data

```

```

def get_reviews_admin():
    db_file = "db.db"
    conn = None
    data = None
    try:
        sql = f""""SELECT * FROM reviews where moder = 0""""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchall()
        cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()
        return data

def up_id_moder_reviews(id_reviews):
    db_file = "db.db"
    conn = None
    try:
        sql = f""""UPDATE reviews SET moder = 1 WHERE id = "{id_reviews}" """"
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        conn.commit()

```

```

        cur.close()
    except Exception as e:
        print(e, 6)
    finally:
        if conn is not None:
            conn.close()

#-----WORK-----
def set_works(_text):
    db_file = "db.db"
    conn = None
    try:
        sql = """INSERT INTO work(text_work,moder) VALUES(?,?);"""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql, (_text,0))
        conn.commit()
        cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()

def get_work_moder():
    db_file = "db.db"
    conn = None
    data = None
    try:
        sql = f"""SELECT * FROM work WHERE moder = 1"""

```

```

    conn = sqlite3.connect(db_file)
    cur = conn.cursor()
    cur.execute(sql)
    data = cur.fetchall()
    cur.close()
except Exception as e:
    print(e, 1)
finally:
    if conn is not None:
        conn.close()
    return data

def get_work_admin():
    db_file = "db.db"
    conn = None
    data = None
    try:
        sql = f""""SELECT * FROM work WHERE moder=0 """
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        data = cur.fetchall()
        cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()

```

```
    return data
```

```
def up_id_moder_work(id_work):
    db_file = "db.db"
    conn = None
    try:
        sql = f"UPDATE work SET moder = 1 WHERE id = \"{id_work}\""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql)
        conn.commit()
        cur.close()
    except Exception as e:
        print(e, 6)
    finally:
        if conn is not None:
            conn.close()
```

```
#-----NEWS-----
```

```
def get_news():
    db_file = "db.db"
    conn = None
    data = None
    try:
        sql = f"SELECT * FROM news"
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
```

```

        cur.execute(sql)
        data = cur.fetchall()
        cur.close()
except Exception as e:
    print(e, 1)
finally:
    if conn is not None:
        conn.close()
    return data
def set_news(_text,_photo):
    db_file = "db.db"
    conn = None
    try:
        sql = """INSERT INTO news(text_news, id_photo) VALUES(?,?);"""
        conn = sqlite3.connect(db_file)
        cur = conn.cursor()
        cur.execute(sql, (_text,_photo))
        conn.commit()
        cur.close()
    except Exception as e:
        print(e, 1)
    finally:
        if conn is not None:
            conn.close()
create_tables()

```

A.3 Компонент markup.py

```

from telebot import types
import db_cmd

def gen_mainmenu():
    markup = types.InlineKeyboardMarkup(row_width=2)
    btn1 = types.InlineKeyboardButton(
        "🛒 Купить", callback_data="buy")
    btn10 = types.InlineKeyboardButton(
        "🛒 Продать", callback_data="sall")
    btn2 = types.InlineKeyboardButton(
        "✍️ Отзывы", callback_data="reviews")
    btn3 = types.InlineKeyboardButton(
        "✂️ Вакансии", callback_data="work")
    btn4 = types.InlineKeyboardButton(
        "📰 Новостная лента", switch_inline_query_current_chat="news")
    btn6 = types.InlineKeyboardButton(
        "👤 Чат", url='https://t.me/joinchat/7mdmivmckJg0ODUy')
    btn5 = types.InlineKeyboardButton(
        "🛠️ Поддержка проекта", callback_data="help")
    btn9 = types.InlineKeyboardButton(
        "🔔 Подписаться на уведомления ", callback_data="scrib")
    markup.add(btn1, btn10)
    markup.add(btn2, btn3)
    markup.row(btn6, btn4)
    markup.row(btn5)
    markup.row(btn9)
    return markup

def set_news():
    markup = types.InlineKeyboardMarkup(row_width=2)
    btn3 = types.InlineKeyboardButton(
        "🌐 Главное меню", callback_data="start")
    markup.row(btn3)
    return markup

def set_img():
    markup = types.InlineKeyboardMarkup(row_width=2)
    btn2 = types.InlineKeyboardButton(
        "⬅️ BACK Назад", callback_data="ad")
    btn3 = types.InlineKeyboardButton(
        "🌐 Главное меню", callback_data="start")

```

```

    markup.row(btn2, btn3)
def get_add():
    markup = types.InlineKeyboardMarkup(row_width=2)
    btn1 = types.InlineKeyboardButton(
        "Да",callback_data="yes")
    btn2 = types.InlineKeyboardButton(
        "Нет",callback_data="not")
    btn3 = types.InlineKeyboardButton(
        "Главное меню",callback_data="start")
    markup.add(btn1,btn2)
    markup.row(btn3)
    return markup

def gen_adminmenu():
    markup = types.InlineKeyboardMarkup(row_width=3)
    btn1 = types.InlineKeyboardButton(
        "Модерация отзывов", callback_data="adminreviews")
    btn2 = types.InlineKeyboardButton(
        "Публикация Новостей", callback_data="adminnews")
    btn3 = types.InlineKeyboardButton(
        "Модерация объявлений", callback_data="moder_sall")
    btn4 = types.InlineKeyboardButton(
        "Модерация вакансий", callback_data="moder_work")
    btn5 = types.InlineKeyboardButton(
        "🌐 Главное меню", callback_data="start")
    markup.add(btn1)
    markup.add(btn3)
    markup.row(btn4)
    markup.row(btn2)
    markup.row(btn5)
    return markup
#-----WORK-----
def gen_work():
    markup = types.InlineKeyboardMarkup(row_width=2)
    btn1 = types.InlineKeyboardButton(
        "Найти вакансию", callback_data="seach_work")
    btn2 = types.InlineKeyboardButton(
        "Опубликовать вакансию", callback_data="add_work")
    btn3 = types.InlineKeyboardButton(
        "Главное меню",callback_data="start")
    markup.add(btn1,btn2)

```

```

markup.row(btn3)
return markup
#-----BUY-----
def categor():
    markup = types.InlineKeyboardMarkup(row_width=2)
    btn1 = types.InlineKeyboardButton(
        "Кальяны",switch_inline_query_current_chat=" hookah/")
    btn2 = types.InlineKeyboardButton(
        "Чаши",switch_inline_query_current_chat=" bowls/")
    btn3 = types.InlineKeyboardButton(
        "Табак",switch_inline_query_current_chat=" tobac/")
    btn4 = types.InlineKeyboardButton(
        "Колбы",switch_inline_query_current_chat=" colbs/")
    btn5 = types.InlineKeyboardButton(
        "Акссесуары",switch_inline_query_current_chat=" other/")
    btn6 = types.InlineKeyboardButton(
        "Главное меню",switch_inline_query_current_chat="start")
    markup.add(btn1,btn2)
    markup.add(btn3,btn4)
    markup.row(btn5)
    markup.row(btn6)
    return markup

def categor_sale():
    markup = types.InlineKeyboardMarkup(row_width=2)
    btn1 = types.InlineKeyboardButton(
        "Кальяны",callback_data="hookah")
    btn2 = types.InlineKeyboardButton(
        "Чаши",callback_data="bowls")
    btn3 = types.InlineKeyboardButton(
        "Табак",callback_data="tobac")
    btn4 = types.InlineKeyboardButton(
        "Колбы",callback_data="colbs")
    btn5 = types.InlineKeyboardButton(
        "Акссесуары",callback_data="other")
    btn6 = types.InlineKeyboardButton(
        "Главное меню",callback_data="start")
    markup.add(btn1,btn2)
    markup.add(btn3,btn4)
    markup.row(btn5)
    markup.row(btn6)
    return markup

```

```
def get_reviews():  
    markup = types.InlineKeyboardMarkup(row_width=2)  
    btn1 = types.InlineKeyboardButton(  
        "Добавить отзыв",callback_data="add_reviews")  
    btn2 = types.InlineKeyboardButton(  
        "Посмотреть отзывы",callback_data="prt_reviews")  
    markup.add(btn1,btn2)  
    return markup
```

ДОДАТОК Б
Слайди презентації

Національний університет “Запорізька політехніка”
Кафедра ПЗ

Випускна кваліфікаційна робота бакалавра на тему:
Розробка телеграм бота з використанням API та SQLite

Виконав:
ст. гр. КНТ-218

І.Д. Батюкін

Керівник:
к.т.н., доцент

О.М. Гладкова

1

Рисунок Б.1 – Слайд 1

МЕТА РОБОТИ

Об’єкт дослідження - програмні засоби для кальянної індустрії.

Предмет дослідження - бот в Telegram для об'єднання аудиторії, що цікавляться кальянною індустрією.

Мета роботи - створення бота в телеграм для кальянної промисловості з використанням бази даних SQLite.

2

Рисунок Б.2 – Слайд 2

ДЛЯ ДОСЯГНЕННЯ ПОСТАВЛЕНОЇ МЕТИ НЕОБХІДНО ВИКОННАТИ НАСТУПНІ ЗАДАЧІ

- Дослідити предметну область;
- Виконати аналіз процесу розробки боту;
- Розробити структуру пояснювальної записки;
- Виконати програмну реалізацію проекту;
- Провести тестування програмного продукту.

3

Рисунок Б.3 – Слайд 3

ВИДИ TELEGRAM БОТІВ

- 1) Магазин - цифрова вітрина товарів.
- 2) Функціональний - чудово виконує одну чи кілька функцій, наприклад бот контролю підписок, бот скачування відео чи парсингу сторонніх ресурсів.
- 3) Криптовалютний - різні гаманці, обмінники та багато іншого.
- 4) Доставка - дозволяє створювати посилки та відстежувати їх
- 5) Консультанти - відповідає на запитання користувачів із заготовленим скриптом або розсилає інформацію для користувачів.

4

Рисунок Б.4 – Слайд 4

ПЕРЕВАГИ ТА НЕДОЛІКИ МОВИ ПРОГРАМУВАННЯ PYTHON

Переваги	Недоліки
Python - це кроссплатформенний, підходить для різних платформ.	Повільніше порівняно з низькорівневий мовами
Спрощений синтаксис та інтуїтивно зрозумілі функції.	Пам'ять Витрати.
Має велику аудиторію.	Мобільний розвиток.
А також багато великих компаній, підтримують Python, при написанні API до своїх проектів.	
Є реалізація інтерпретаторів для мобільних пристроїв та непопулярних систем	

5

Рисунок Б.5 – Слайд 5

ВИБІР СЕРЕДОВИЩА РОЗРОБКИ:

	Переваги	Недоліки
Eclipse + PyDev	Якщо у вас вже було встановлено Eclipse, то установка PyDev пройде швидко та гладко. У досвідченого користувача Eclipse не виникне проблем із вивченням цього розширення.	Якщо ви починаєте вивчати Python або розробку в цілому, Eclipse може стати непосильною ношею. Складний у налаштуванні.
Sublime Text 3	Швидкий, легкий та має гарну підтримку. Мінімалістичний дизайн, підтримка різних операційних систем.	Sublime Text не є безкоштовним, хоча ви можете використовувати пробний період скільки завгодно. Установка розширень може перетворитися на ще квест. Крім того, у редакторі немає підтримки налагодження та запуску коду.
Atom	Підтримка на всіх платформах завдяки Electron. Atom легко і швидко скачується та завантажується.	Підтримка збирання та налагодження не вбудована, а додається за допомогою розширень. Оскільки Atom написаний за допомогою Electron, він завжди працює як JavaScript-процес, а не як нативна програма.

6

Рисунок Б.6 – Слайд 6

ВИБІР БАЗ ДАНИХ:

	Переваги	Недоліки
MongoDB	Схема менша. Якщо у вас є гнучка схема, це ідеально підходить для сховища документів. Легкість масштабування. Масштабування записується за допомогою функції sharding. MongoDB є безкоштовним і може працювати на Linux.	Розмір даних у MongoDB зазвичай вищий через, наприклад, кожен документ має імена полів, які зберігаються в ньому. Менше гнучкості при запиті (наприклад, немає JOIN). Немає підтримки транзакцій.
PostgreSQL	Легко використати. Має тип даних користувача. Відкрите джерело. Велика підтримка спільноти. Використовуйте процедури, що зберігаються.	Створює окремий сервіс для кожного клієнта, який перетворюється на велике використання пам'яті. Не дуже гарний у плані продуктивності. Робити реплікацію складніше. Встановлення не <u>легке</u> для <u>новачка</u> .
SQLite 3	Легко використати. Має тип даних користувача. Відкрите джерело. Велика підтримка спільноти. Використовуйте процедури, що зберігаються.	Створює окремий сервіс для кожного клієнта, який перетворюється на велике використання пам'яті. Не дуже гарний у плані продуктивності. Робити реплікацію складніше. Встановлення не <u>легке</u> для <u>новачка</u> .

7

Рисунок Б.7 – Слайд 7

ЗАГАЛЬНА СТРУКТУРА БОТУ



8

Рисунок Б.8 – Слайд 8

ТАБЛИЦІ БАЗИ ДАНИХ

Назва таблиці	Опис таблиці
<u>users</u>	Зберігає в собі інформацію про всіх користувачів бота
<u>product</u>	Зберігає в собі інформацію про товари які виставлені на продаж або чекають на модерацію
<u>users_admin</u>	Зберігає в собі інформацію про адміністрацію робота
<u>news</u>	Зберігає інформацію про новини, які опубліковані
<u>reviews</u>	Зберігає у собі всі відгуки, які опубліковані або чекають на модерацію
<u>work</u>	Зберігає у собі всі вакансії, які опубліковані або чекають на модерацію

9

Рисунок Б.9 – Слайд 9

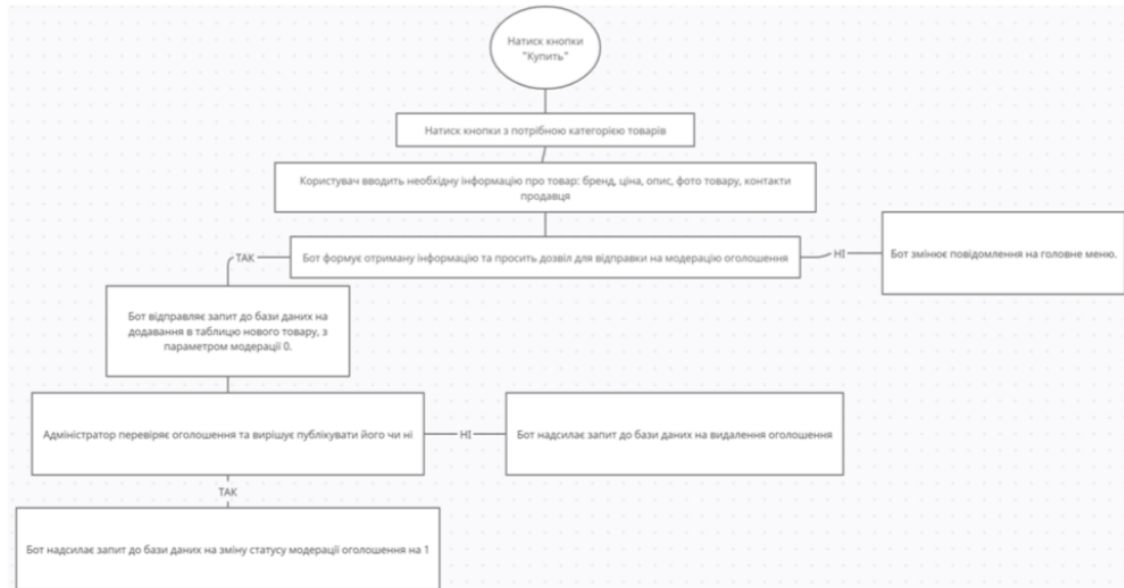
СХЕМИ АЛГОРИТМІВ РОБОТИ З БОТОМ: ПОКУПКИ



10

Рисунок Б.10 – Слайд 10

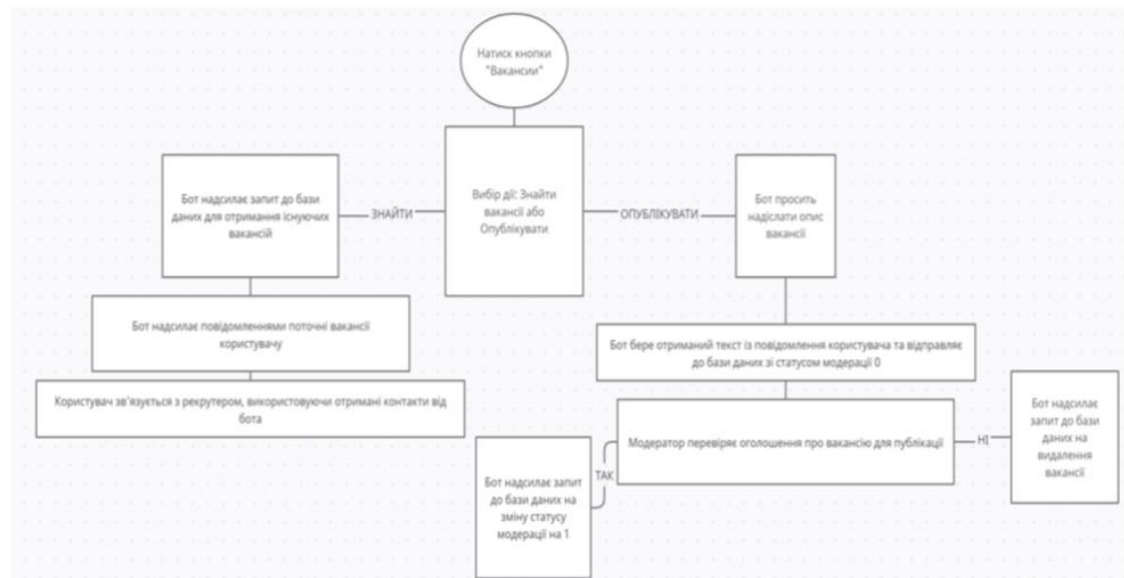
СХЕМИ АЛГОРИТМІВ РОБОТИ З БОТОМ:ПРОДАЖИ



11

Рисунок Б.11 – Слайд 11

СХЕМИ АЛГОРИТМІВ РОБОТИ З БОТОМ:ВАКАНСІЇ



12

Рисунок Б.12 – Слайд 12

ДЕМОНСТРАЦІЯ РОБОТИ БОТА

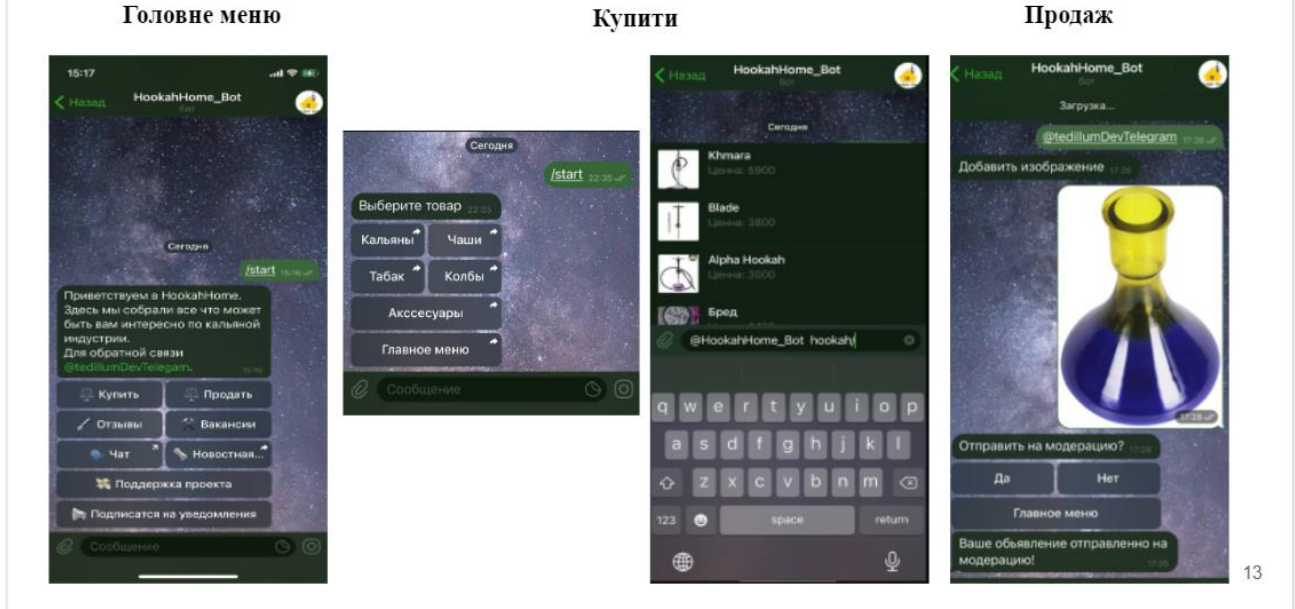


Рисунок Б.13 – Слайд 13

ВИСНОВКИ

- У ході експлуатації критичних помилок не було виявлено.
- Бот дозволяє знайти всю цікаву інформацію для користувача в сфері кальянів, але планується подальший розвиток продукту. Буду покращуватися виведення відгуків і вакансій з подальшим додаванням фільтра. Також буде покращено фільтрацію оголошень для більш комфортного пошуку.
- Буде доданий ще блок порад, де люди зможуть залишати свої поради щодо тематики кальянів. Планується зі зростанням аудиторії, додаванням рекламних постів та можливості просування своїх оголошень.
- На даному етапі є унікальний продукт, аналога, якому немає. Реалізований інтерфейс легко та пропонує найнеобхідніше для користувача.

Рисунок Б.14 – Слайд 14