Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Юрій КРАВЧЕНКО
«_____»_____2022 року

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»
освітньо-професійна програма «Мережеві та інтернет технології»

на тему:

РОЗРОБКА ІНТЕРАКТИВНОГО СЕРВІСУ ОБРОБКИ ЗАМОВЛЕНЬ З ВИКОРИСТАННЯМ ТЕЛЕГРАМ-БОТІВ

Виконав: студент групи МІТ -41

Ханкішиєв Віктор Халідович _____
(прізвище ім'я по-батькові) (підпис)

Керівник: професор кафедри мережевих та інтернет технологій

Дуднік Андрій Сергійович _____
(прізвище ім'я по-батькові) (підпис)

Київ 2022

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра мережевих та інтернет технологій

ЗАТВЕРДЖУЮ
завідувач кафедри
мережевих та інтернет технологій
_____ Ю.В. Кравченко
«_____»_____2021 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ

Здобувачу вищої освіти Ханкішиєв Віктор Халідович
(прізвище, ім'я, по батькові)

1. Тема роботи:

«Розробка інтерактивного сервісу обробки замовлень з використанням телеграм-ботів»

затверджена на засіданні кафедри МІТ «24» грудня 2021 р. протокол № 8
2. Термін здачі закінченої роботи «30» травня 2022 р.
3. Вихідні дані до проекту (роботи)

4. Зміст пояснювальної записки (перелік питань, що їх потрібно розробити, обсяг – 35-50 стор.)

1. Дослідження підходів та інструментів які використовуються для створення телеграм ботів.
2. Проектування інтерактивного сервісу обробки замовлень.
3. Реалізація інтерактивного сервісу обробки замовлень

5. Перелік графічного матеріалу 8-10 слайдів

Актуальність теми дипломної роботи.
Вибір інструментів для реалізації сервісу.
Створення сервісу.
Оцінка ефективності сервісу.
Висновки.

Дата видачі завдання
Керівник роботи Дуднік А.С.
(підпис) (посада, прізвище, ім'я, по батькові)

Завдання прийняв до виконання Ханкішиєв В. Х.
(підпис) (прізвище, ім'я, по батькові)

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Підготовчий	01.02.2022	
2	Розділ 1	01.03.2022	
3	Розділ 2	01.04.2022	
4	Розділ 3	15.05.2022	
5	Доповідь та слайди	30.05.2022	
6	Пояснювальна записка	30.05.2022	

Здобувач вищої освіти _____ Ханкішиєв В. Х.
(підпис) (прізвище, ім'я, по батькові)

Керівник _____ Дуднік А. С.
(підпис) (прізвище, ім'я, по батькові)

РЕФЕРАТ

Пояснювальна записка: 85 с., 7 рис., 6 табл., 18 джерел.

Об’єкт дослідження: обробка замовлень у комерційній чи громадській організації.

Предмет дослідження: процес взаємодії замовника та виконавця під час обробки замовлення.

Мета роботи (проєкту): розробити телеграм ботів для полегшення процесу обробки замовлень як зі сторони користувача, так і зі сторони виконавця замовлень.

Методи дослідження: системний підхід.

У роботі проведено аналіз існуючих рішень у сфері створення та обробки замовлень, їх ефективності.

Запропоновано розробити сервіс з обробки замовлень, який би дозволяв забезпечити інтерактивну співпрацю між замовником та виконавцем замовлення.

Побудовано логічну та фізичну моделі даних бази даних, яка вміщатиме дані по замовленню, його стадії та іншу.

Розроблено телеграм-ботів для полегшення процесу обробки замовлень як зі сторони користувача, так і зі сторони виконавця замовлень.

Практичне значення роботи полягає у наданні невеликим організаціям, благодійним фондам та волонтерським організаціям зручного сервісу з обробки замовлень.

Результати здійснених у дипломному проєкті досліджень можуть бути використані невеликими організаціями, благодійними фондами та волонтерськими організаціями для спрощеної комунікації з клієнтами та полегшення підтримки замовлень.

Напрямки подальших досліджень роботи включають інтеграцію розробленого сервісу у роботу організацій, які надають послуги у відповідній сфері.

Ключові слова:

ЗАМОВЛЕННЯ, СЕРВІС, РЕЛЯЦІЙНА БАЗА ДАНИХ, ТЕЛЕГРАМ-БОТ, SQLITE, TELEGRAM BOT API, PYTHON

ЗМІСТ

Стор.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 9
ВСТУП 9
РОЗДІЛ І. АНАЛІЗ ЗАДАЧІ РЕАЛІЗАЦІЇ ІНТЕРАКТИВНОГО СЕРВІСУ ОБРОБКИ ЗАМОВЛЕНЬ З ВИКОРИСТАННЯМ ТЕЛЕГРАМ-БОТІВ 10
1.1 Дослідження підходів та інструментів, які використовуються для обробки замовлень 10
1.1.1. Бот, як засіб для інтерактивного сервісу обробки замовлень 11
1.1.2. Проблеми обробки замовлень 13
1.2 Постановка задачі 14
1.3 Вибір технологій та інструментів для реалізації інформаційної технології 15
1.3.1 Бази даних 15
1.3.2 Платформи 18
РОЗДІЛ ІІ. ПРОЄКТУВАННЯ ІНТЕРАКТИВНОГО СЕРВІСУ ОБРОБКИ ЗАМОВЛЕНЬ 20
2.1 Проєктування бази даних 20
2.1.1 Вибір СУБД 20
2.1.2 Проєктування моделей 20
2.2 Проєктування інтерфейсу 25
2.2.1 Вибір платформи 25
2.2.2 Функції сервісу 26
2.2.3 Вибір мови програмування 27
Висновки по розділу 2 29
РОЗДІЛ ІІІ. РЕАЛІЗАЦІЯ ІНТЕРАКТИВНОГО СЕРВІСУ ОБРОБКИ ЗАМОВЛЕНЬ 30
3.1 Створення бази даних 30
3.1.1 Створення таблиць 30
3.2 Розробка програмного додатка 32
3.3 Взаємодія користувача з сервісом 37
Висновки по розділу 3 40
ВИСНОВОК 41
ПЕРЕЛІК ПОСИЛАНЬ 42

ДОДАТОК А 44

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІТ – інформаційні технології.

ІС – інформаційна система.

БД- база даних.

СУБД – система управління базами даних.

РБД – реляційна база даних.

РСУБД – реляційна система управління базами даних.

DML – Data Manipulation Language (мова маніпулювання даними).

DDL – Data Definition Language (мова опису даних).

ООМД – об’єктно-орієнтована модель даних.

ООП – об’єктно-орієнтоване програмування.

ООБД – об’єктно-орієнтована база даних.

ОРСУБД – об’єктно-реляційна СУБД.

ОРБД – об’єктно-реляційна база даних.

СУРБД – системи управління розподіленої базою даних.

ПК – персональний комп’ютер.

ОС – операційна система.

ВСТУП

Спочатку COVID-19 повністю змінив світ, тепер наша країна знаходиться у стані війни. Світ швидко змінюється і деякі процеси потрібно значно пришвидшувати. Наразі багато невеликих підприємств та організацій реабілітуються та намагаються повернутись до нормального робочого графіку, однак, для багатьох з них найкращою опцією було б уникнути великих скупчень у приміщеннях та централізованого місцезнаходження. Наразі дана модель роботи не є найкращою альтернативою, оскільки постійне відвідування місць роботи створює ризик для безпеки робітників.

Тому актуальним питанням є те, чи можна змінити своє відношення до централізації робочого офісу на постійного місцезнаходження співробітників.

Насамперед актуальність даного питання легко перевіряється тим, яка кількість людей поїхала в інші міста або країни жити та працювати і, наприклад, кол-центри для прийому замовлень більше не працюватимуть настільки стабільно, як раніше.

Технології не стоять на місці і постійно розвиваються, багатьом людям не подобається витрачати час на живе спілкування, тому вже недостатньо просто мати кол-центр, в якому співробітники прийматимуть замовлення. Але в умовах невеликої організації з обмеженим бюджетом не завжди є можливість створити та підтримувати сайт з гарним інтерфейсом та великою базою даних, або навіть утримувати співробітників у кол-центрах.

Розробка даного сервісу покликана спростити виконання задач по прийому та обробці замовлень для організацій, які не мають великих можливостей та необмеженого бюджету. Актуальною дана розробка також може стати громадських організаціям та благодійним фондам, які надають певні послуги на безоплатній основі.

РОЗДІЛ І. АНАЛІЗ ЗАДАЧІ РЕАЛІЗАЦІЇ ІНТЕРАКТИВНОГО СЕРВІСУ ОБРОБКИ ЗАМОВЛЕНЬ З ВИКОРИСТАННЯМ ТЕЛЕГРАМ-БОТІВ

1.1 Дослідження підходів та інструментів, які використовуються для обробки замовлень

На даний момент майже кожне підприємство має засіб, за допомогою якого можна хоча б переглянути перелік їх товарів, якщо не замовити, оскільки, починаючи з другої половини другого десятиліття XXI століття для кожної організації сайт став невід'ємним атрибутом. Це значно спрощує існування різноманітних підприємств, а в деяких випадках це дозволяє навіть відмовитись від звичної форми магазину, не витрачати додаткових коштів на персонал та оренду і не тримати ніяких приміщень, окрім складу, що дозволяє магазину перейти повністю в онлайн формат. Якщо замислитись над тим, який підхід більш раціональний, то варто проаналізувати, які зараз існують технології для обробки замовлень.

На даний момент шаблони, які застосовуються під час організації підприємства, яке щось продає, або постачає можна поділити на наступні категорії:

- організації (магазини, фонди), в яких фізично знаходяться співробітники і нема ніякого інтернет-ресурсу, де можна хоча б переглянути асортимент (Звичайні супермаркети);
- організації (магазини, фонди), в яких фізично знаходяться співробітники, але наявний інтернет ресурс, де можна переглянути асортимент, або зробити онлайн-замовлення (такі підприємства, як наприклад «Розетка», «Фокстрот» та інші);

– організації (магазини, фонди), які працюють повністю в онлайн і не мають фізичного місцезнаходження, окрім офісу, або без нього (такі як магазини в соціальних мережах).

Також варто розглянути інструменти, які організації можуть використовувати для часткового, або повного переходу в онлайн:

– веб-сайт;
– бот у месенджері;
– мобільний додаток.

Виходячи з того, що розглядається можливість використання даних методів для організацій з невеликим або дуже малим бюджетом та невеликим переліком товарів, послуг, або взагалі без нього – можна відмовитись від додатку та відмовитись від сайту, оскільки витрати на веб-сайт будуть значно більші, ніж на створення та підтримку бота.

1.1.1. Бот, як засіб для інтерактивного сервісу обробки замовлень

Боти в месенджерах – зручне рішення, вони не потребують постійного підключення працівників з боку організації, це цілком автономне та самостійне технічне рішення, яке можна налаштувати, так, як заманеться – наприклад, надсилати повідомлення працівнику, коли з’являється нове замовлення, або навпаки – лишити процес повністю автономним і давати перевіряти нові замовлення в будь-який момент. Хоча боти це і досить нова сфера, але в Україні це вже досить поширена річ для служб підтримки багатьох організацій, деякі компанії переносять повноцінний функціонал обробки замовлень, зв’язку з підтримкою і так далі до боту. Розглянемо можливі додатки для реалізації бота. Зазвичай, це так звані месенджери, тобто спеціалізовані додатки, які дозволяють обмінюватися повідомленнями.

1) Telegram.

Telegram став одним з найпопулярніших месенджерів в Україні майже з моменту його появи. На момент його виходу на ринок функціонал в ньому був значно кращий, як і синхронізація даних користувачів, ніж у Viber на той момент і це стало вирішальним фактором в успіху Telegram на просторах України. На даний момент це один з найпопулярніших месенджерів на рівні з Viber та Facebook Messenger. Значною перевагою Telegram є те, що його API є повністю безкоштовним та доступним для всіх, забезпечене детальною документацією. Реалізувати можна як найпростіших ботів, так і ботів, що, наприклад, мають інтеграції з криптовалютними гаманцями, обробкою банківської інформації та інше.

2) Viber.

Ще один достатньо популярний месенджер в Україні. За функціоналом та зручністю користування він трохи поступається Telegram, але це не робить його менш популярним. Має нетиповий функціонал для створення ботів – можна зробити власний дизайн кнопок, шпалер.

3) Facebook Messenger.

Facebook Messenger – очевидно, один з найпопулярніших у світі месенджерів, оскільки напряму є частиною функціоналу Facebook і являється окремим додатком для обміну повідомленнями користувачів. Не дивлячись на те, що даний месенджер має, мабуть, найбільшу аудиторію і до якого найпростіше відправити користувача через рекламу у Facebook – бот в даному месенджері не є найкращим рішенням, оскільки у ботів в Facebook немає можливості надсилати push-сповіщення користувачам, а це означає, що людина може просто не побачити, що її замовлення, наприклад, відмінили, або перенесли дату доставки і тому подібне.

4) WhatsApp.

Даний месенджер не є дуже популярним в Україні. І, хоча у Європі це, мабуть, найпопулярніший месенджер після Facebook Messenger, WhatsApp має один величезний недолік – його API платний. І навіть можливість сплатити доступ до API не гарантує того, що доступ буде отриманий,

оскільки організація має подати заявку, обґрунтувати потребу у створені бота і після цього адміністрація WhatsApp прийматиме рішення – надавати доступ, чи ні.

1.1.2. Проблеми обробки замовлень

Потрібно розуміти, що на даний момент життя вкрай нестабільне, кожного дня відбувається щось нове і неможливо спрогнозувати, яким буде завтра, тому перехід до онлайн моделі обробки замовлень без постійної безпосередньої участі людини зі сторони виконавця є оптимальним.

Сучасні технології дозволяють виключити з процесу створення замовлення працівника компанії, так само сучасні технології дозволяють виключити потребу в безпосередньому контакті працівника організації з замовником при зміні статусу замовлення. Також, на даний момент є можливість позбавитись телефонного спаму від людей, які мають за ціль не зробити замовлення, а нашкодити організації, оскільки простіше закрити доступ за допомогою бази даних такому користувачу, аніж приймати кожного разу телефонний виклик.

Також, подібний підхід прибирає можливість давати взаємодіяти з користувачами несумлінним працівникам, які можуть недоброзичливо спілкуватись з клієнтами, фактично, від імені організації, спираючись на власний психологічний стан, настрій і тому подібні фактори.

Отже, автоматизація даних процесів є найраціональнішим рішенням, оскільки користувач може створювати замовлення, дізнаватись стадію його виконання та скасовувати замовлення в неробочі години організації, чим пришвидшує процес, і, так само, організація може повідомляти клієнта про

зміну стадії виконання замовлення, його відміну, або прийняття в обробку в будь який момент, без нагальної потреби чекати, поки клієнт підніме слухавку та поспілкується зі співробітником організації.

1.2 Постановка задачі

Найелементарніший процес створення замовлення зазвичай являє собою діалог, в якому клієнт описує нагальну потребу, а виконавець обробляє її і віддає результат. На елеметраному рівні – це придбання чого-небудь з полиці магазину. Але в даному випадку до створення інтерактивного сервісу обробки замовлень необхідно підходити до цього питання з наукової точки зору.

Предметна область – дані про користувача, співробітника та замовлення, які відображаються у БД.

Модель предметної області – знання, що існують про предметну область. Модель описує ті процеси, що відбуваються в предметній області, а також дані, які являють собою предметну область. Від коректного опису будови предметної області залежить успіх реалізації сервісу.

Як правило, модель предметної області будується на трьох рівнях:

- зовнішній рівень. На цьому рівні визначають вимоги, які саме є потреби від сервісу. Визначаються елементи сервісу – об’єкти;
- концептуальний рівень. На цьому рівні визначають, яким чином взаємодіють між собою компоненти і об’єкти системи, як функціонує сервіс;
- внутрішній рівень. На цьому рівні вирішується, яким чином та за допомогою яких програмних і технічних засобів усі вимоги повинні реалізовуватися.

Фактично, сервіс має два окремих боти та БД, які є неподільною предметною областю, оскільки їх існування без одного з трьох членів не має

жодного сенсу. Боти можуть існувати без БД, але не буде куди записувати інформацію, так само можуть існувати бот і БД, але без одного з ботів не буде клієнтів, які створюють замовлення, а без іншого не буде можливості переглянути створені замовлення та взаємодіяти з ними. Сервіс має бути простим у використанні навіть для нових і зовсім недосвідчених у використанні новітніх технологій користувачів.

У боті для користувачів має бути можливість реєструватись, швидко створювати замовлення, переглядати їх і отримувати повідомлення, коли їхнє замовлення відміняють або приймають в обробку. В боті для сторони виконавця замовлень, а в конкретному випадку – волонтерського фонду, має бути можливість переглядати нові замовлення користувачів, приймати їх до виконання, скаржитись на замовлення, що не несуть в собі ніякого сенсу, або є «спамом», а також завершувати замовлення або відмовлятись від їх виконання.

Головним чином сервіс повинен допомагати користувачам швидко та просто створювати замовлення, а волонтерам переглядати їх та швидко реагувати на них відповідним чином, забезпечувати зручну комунікацію між замовником та виконавцем у процесі обробки замовлення.

1.3 Вибір технологій та інструментів для реалізації інформаційної технології

1.3.1 Бази даних

Дані – це сукупність окремої одиниці інформації. Говорячи з точки зору обчислень, дані – це в основному інформація, яка може бути перетворена в певну форму для ефективного переміщення та обробки. Наприклад: дані замовлення, регіон замовлення, ім'я та прізвище користувача тощо.

База даних – це організований набір структурованих даних, що робить БД легкодоступною, керованою та оновлюваною. Простими словами, база даних – це місце, де зберігаються дані.

Наявні три основні типи баз даних: ієрархічні, мережеві та реляційні. Ієрархічний тип дуже відрізняється від мережевого та реляційного. В ньому для довільного зв'язку один елемент обирають головним та називають його безпосереднім предком, а інший – підлеглим та називають його безпосереднім нащадком. Об'єкт завжди має не більше одного предка, проте може мати необмежену кількість нащадків, або не мати їх зовсім.

Мережева моделі відрізняється від ієрархічної таким чином – у ній відсутнє підпорядкування. Елементи різних рівнів мережевої моделі вільно зв'язані.

Реляційна модель – це модель, в якій набір даних створюється із заздалегідь визначеними зв'язками, вона побудована на взаємовідношенні даних, що її складають. В цілому, реляційна модель – це сукупність взаємозв’язаних таблиць, кожна з яких містить набір об'єктів. Таблиці такої моделі даних, зазвичай, зв'язують між собою за допомогою зовнішніх ключів. Зовнішній ключ – це поле таблиці, яке посилається на первиннний ключ в іншій таблиці. Переваги реляційної моделі:

- простота розуміння та доступність у використанні;
- незалежність даних;
- суворі правила проєктування;
- для організації запитів достатньо ознайомитись зі структурою БД.

Для створення та обробки БД використовують системи управління базами даних (СУБД).

На даний момент СУБД є досить велика кількість. Серед десятки найпопулярніших є Oracle, MySQL, PostgreSql, SQLite. Проведемо аналіз вказаних СУБД.

Oracle – об'єктно-реляційна система керування базами даних від Oracle Corporation. До переваг даної СУБД належать:

- підтримка найбільших за розміром БД;
- централізована система управління та контролю;
- швидкість обробки транзакцій.

Oracle має також і свої недоліки:

- висока вартість ліцензії на комерційне використання;
- СУБД може вимагати великої кількості ресурсів.

MySQL – це найпоширеніша повноцінна серверна СУБД. Перевагами MySQL є:

- простота в роботі;
- масштабованість;
- багатий функціонал.

Недоліки MySQL:

- Недостатній рівень надійності;
- низька швидкість розробки.

SQLite, на відміну від інших СУБД не використовує архітектуру клієнт-сервер. SQLite зберігає всі дані БД в одному файлі, якій можна легко переносити між платформами. Завдяки цьому БД, створена за допомогою SQLite інтегрується напряму в програму і відпадає необхідність у автономних процесах.

Переваги SQLite:

- безкоштовне ПЗ;
- простота у використанні;
- мала ресурсозатратність;
- БД легко переноситься та легко створюються резервні копії;
- наявні засоби захисту від пошкоджень файлів БД;
- 100% покриття тестами коду SQLite.

Недоліки SQLite:

– відсутність системи користувачів;
– відсутність можливості збільшити продуктивність БД.

PostgreSQL є однією з найпоширеніших СУБД. PostgreSQL побудована так, що використовує майже повною мірою ANSI / ISO SQL стандарти своєчасно з виходом нових версій.

Переваги PostgreSQL:

– безкоштовне ПЗ з відкритим вихідним кодом, що відповідає стандарту SQL;
– велика кількість доповнень;
– легке розширення функціоналу;
– об'єктна орієнтованість.

Недоліки PostgreSQL:

– висока вірогідність зменшення швидкості роботи СУБД;
– складний процес налаштування перед початком роботи із СУБД

1.3.2 Платформи

Фактично, вибором платформи для даного проєкту є вибір одного з найпопулярніших месенджерів в Україні на даний момент. Месенджер – це додаток для миттєвого обміну повідомленнями. Боти в месенджерах створюють за допомогою API месенджерів. API – це набір визначень підпрограм та протоколів взаємодії. Фактично, це набір визначених методів для взаємодії компонентів. На даний момент часу можна виділити чотирі основні месенджери, що використовують в Україні:

– Telegram;
– Viber;
– Facebook Messenger;
– WhatsApp.

Проведемо аналіз даних месенджерів:

1. Telegram – месенджер, легкий та зручний у використанні, синхронізація даних користувача виконана на високому рівні, тому перехід між платформами та пристроями для використання даного месенджеру не викликає жодних проблем. За даними опитування ain.ua з досить невеликою вибіркою у 1,5 тисячі чоловік, для повсякденного спілкування Telegram використовують 50,6% опитаних. Значною перевагою Telegram для створення бота є безкоштовний API.

2. Viber – месенджер, другий за популярністю даного опитування, ним для повсякденного спілкування користується 18,5% опитаних респондентів. Також має безкоштовний API.

3. Facebook Messenger – третій за результатами опитування, його використовують 8,1% опитаних. Він має один недолік – ботам у Facebook заборонено використання push-сповіщень. Через даний недолік використання Facebook не є оптимальним.

4. WhatsApp. Найнепопулярніший з перерахованих – 6% з опитаних. Основною проблемою WhatsApp у створенні бота є те, що API для створення ботів є платним. До того ж, гроші не є вирішальним фактором – організація повинна звернутись до WhatsApp з поясненням навіщо потрібен доступ до API, якою є організація, який буде функціонал бота і так далі.

1.4 Висновки по розділу 1

Було розглянуто підходи та інструменти, що використовуються для реалізації інтерактивної системи обробки замовлень, а саме розглянуто бота як засіб для створення системи обробки замовлень, поставлено конкретну задачу, проаналізовано СУБД та платформи для створення ботів.

РОЗДІЛ ІІ. ПРОЄКТУВАННЯ ІНТЕРАКТИВНОГО СЕРВІСУ ОБРОБКИ ЗАМОВЛЕНЬ

2.1 Проєктування бази даних

2.1.1 Вибір СУБД

Серед СУБД була обрана SQLite. SQLite, по-перше, найбільш використовуваний рушій для БД у світі. По-друге, враховуючи особливість проєкту, а саме розрахунок на невеликі організації, – ресурсів БД, що створена одним файлом, буде цілком достатньо. По-третє, використання SQLite дозволяє позбавитись ресурсовитратних фонових процесів. Тому SQLite є найоптимальнішим рішенням для поточної задачі.

2.1.2 Проєктування моделей

ER-модель (entity-relationship model), так звана модель «сутність-зв’язок». Дана модель спрощує побудову БД, оскільки вона визначає основні сутності, їх атрибути та взаємодію між ними. Сутність – це будь який об’єкт, що можна виділити з предметної області, для якої розробляється БД. Наприклад, в даному сервісі сутностями будуть користувач, волонтер та замовлення. Атрибутами сутностей є дані, які БД буде зберігати про кожну сутність, наприклад, атрибутами волонтеру буде його ідентифікаційний номер (id), унікальний ідентифікаційний номер, що присвоює телеграм користувачу (tg_user_id), ім’я (firstname), прізвище (lastname), дата народження (birth_date) та кількість скарг на замовлення користувача (reports).

Етапи створення ER-моделі:

1. Визначення сутностей.

2. Визначення атрибутів.
3. Визначенні унікальних ідентифікаторів.
4. Визначення відношень та їх властивостей.

Сутності предметної області:

1. Користувач.
2. Замовлення.
3. Волонтер (Виконавець).

Атрибути сутностей:

1. Користувач:
   - id
   - tg_user_id
   - firstname
   - lastname
   - birth_date
   - reports
2. Замовлення:
   - order_id
   - order_desc
   - order_date
   - order_status
   - user_id
   - volunteer_id
   - order_urgency_id
   - region_id
   - reports
3. Волонтер:
   - volunteer_id
   - tg_volunteer_id
   - firstname

– lastname

– birth_date

На рисунку 2.1 представлена ER діаграма для даної предметної області.

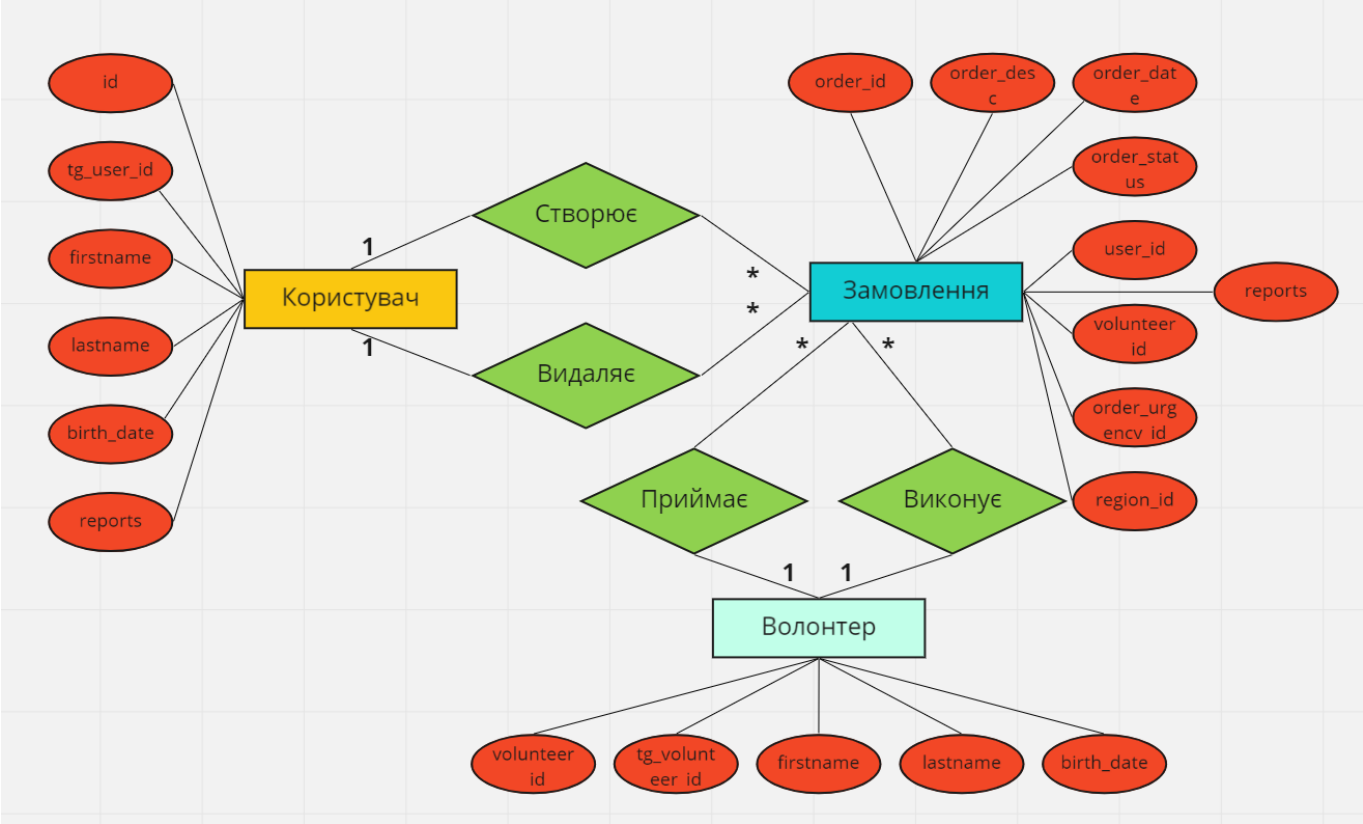

Рисунок 2.1 – ER діаграма БД інтерактивного сервісу обробки замовлень

Також варто розглянути фізичну модель БД. Фізична модель є конкретним описом каталогу даних, отже в ній є інформація про всі об’єкти БД. Побудова фізичної моделі під час проєктування є важливим кроком, оскільки фізична модель будується під конкретну СУБД і показує не тільки атрибути, а ще й їх типи даних. Також фізична модель є дуже важливою через те, що саме на ній базується побудова схеми БД і налаштування взаємодії програми з БД.

На рисунку 2.2 відображена фізична модель БД.

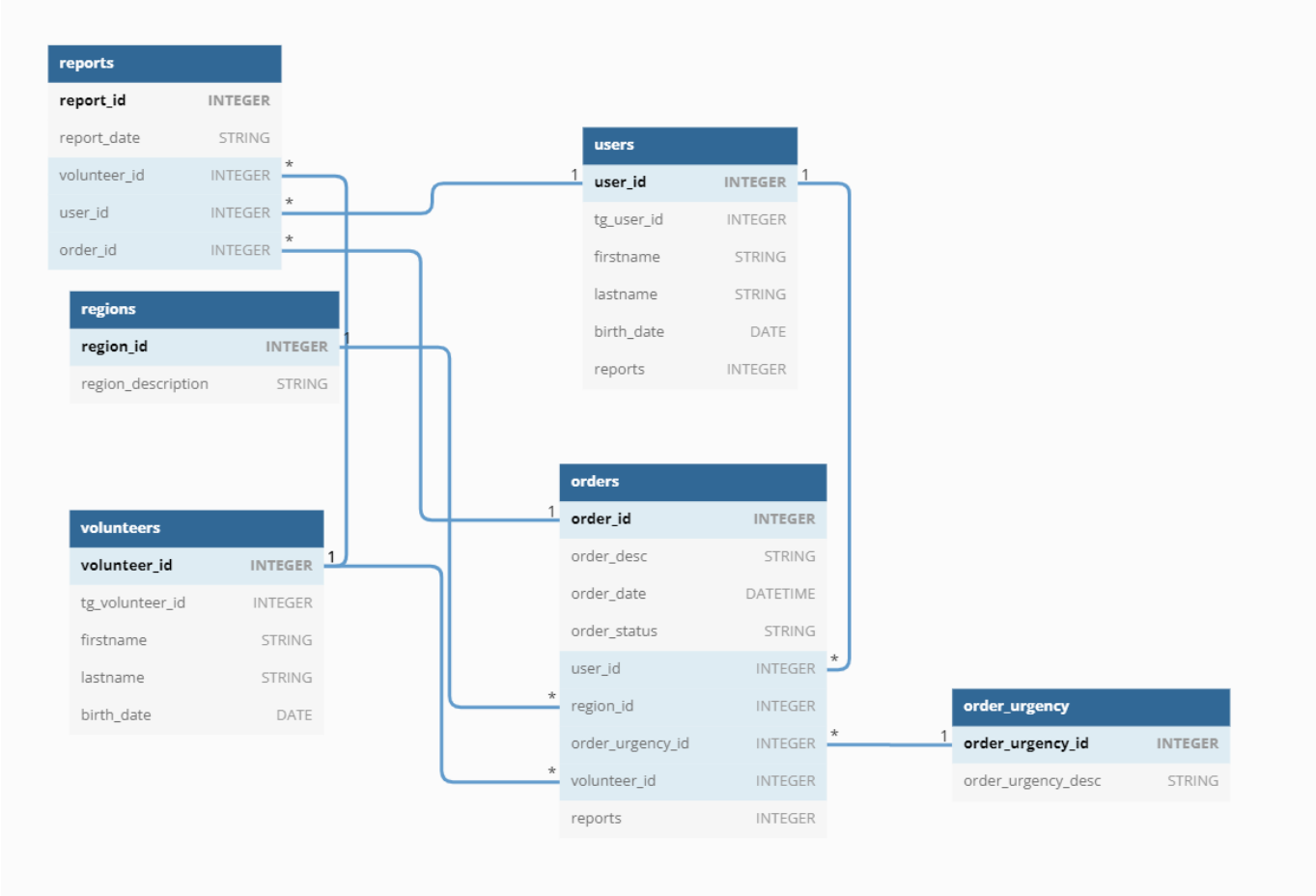

Рисунок 2.2 – Фізична модель БД сервісу

Нижче приведено фізичну схему таблиць БД.

Таблиця 2.1 – Фізична схема таблиці «users/Користувачі»

Атрибут	Властивість
Id	INT, AI, PK
Tg_user_id	INT, UNIQUE
Firstname	STRING
Lastname	STRING
Birth_date	DATE
Reports	INT

Таблиця 2.2 – Фізична схема таблиці «orders/Замовлення»

Атрибут	Властивість
Order_id	INT, AI, PK
Order_desc	STRING
Order_date	DATETIME
Order_status	STRING
User_id	INT, FK
Region_id	INT, FK
Order_urgency_id	INT, FK
Volunteer_id	INT, FK
Reports	INT

Таблиця 2.3 – Фізична схема таблиці «volunteers/Волонтери»

Атрибут	Властивість
Volunteer_id	INT, AI, PK
Tg_volunteer_id	INT, UNIQUE
Firstname	STRING
Lastname	STRING
Birth_date	DATE

Таблиця 2.4 – Фізична схема таблиці «reports/Скарги»

Атрибут	Властивість
Report_id	INT, AI, PK
Report_date	DATETIME
Volunteer_id	INT, FK
User_id	INT, FK
Order_id	INT, FK

Таблиця 2.5 – Фізична схема таблиці «regions/Регіони»

Атрибут	Властивість
Region_id	INT, AI, PK
Region_description	STRING

Таблиця 2.6 – Фізична схема таблиці «order_urgency/Терміновість замовлення»

Атрибут	Властивість
Order_urgency_id	INT, AI, PK
Order_urgency_desc	STRING

При переході від логічної моделі до фізичної було спроєктовано фізичну модель, розкрито зв’язки між сутностями та створено додаткові зв’язки для коректності роботи БД.

2.2 Проєктування інтерфейсу

2.2.1 Вибір платформи

Наразі вибір платформи не є великою проблемою, оскільки в Україні в основному користуються всього чотирьома месенджерами, а саме Telegram, Viber, Facebook та WhatsApp. Оскільки WhatsApp має суттєвий недолік у вигляді платного API, а Facebook має несуттєвий недолік, у вигляді заборони на відправку push-сповіщень від ботів до користувачів, то залишаються тільки Viber та Telegram. Оскільки між ними нема майже ніякої різниці у функціоналі, то цей вибір базується на власному досвіді та тому, що більшість людей в Україні надає перевагу Telegram, а не Viber. Отже, для створення інтерактивного сервісу обробки замовлень було обрано платформу Telegram.

2.2.2 Функції сервісу

Для того, щоб коректно створити сервіс, спочатку потрібно описати бізнес-процес. Для цього створимо діаграму бізнес-процесів.

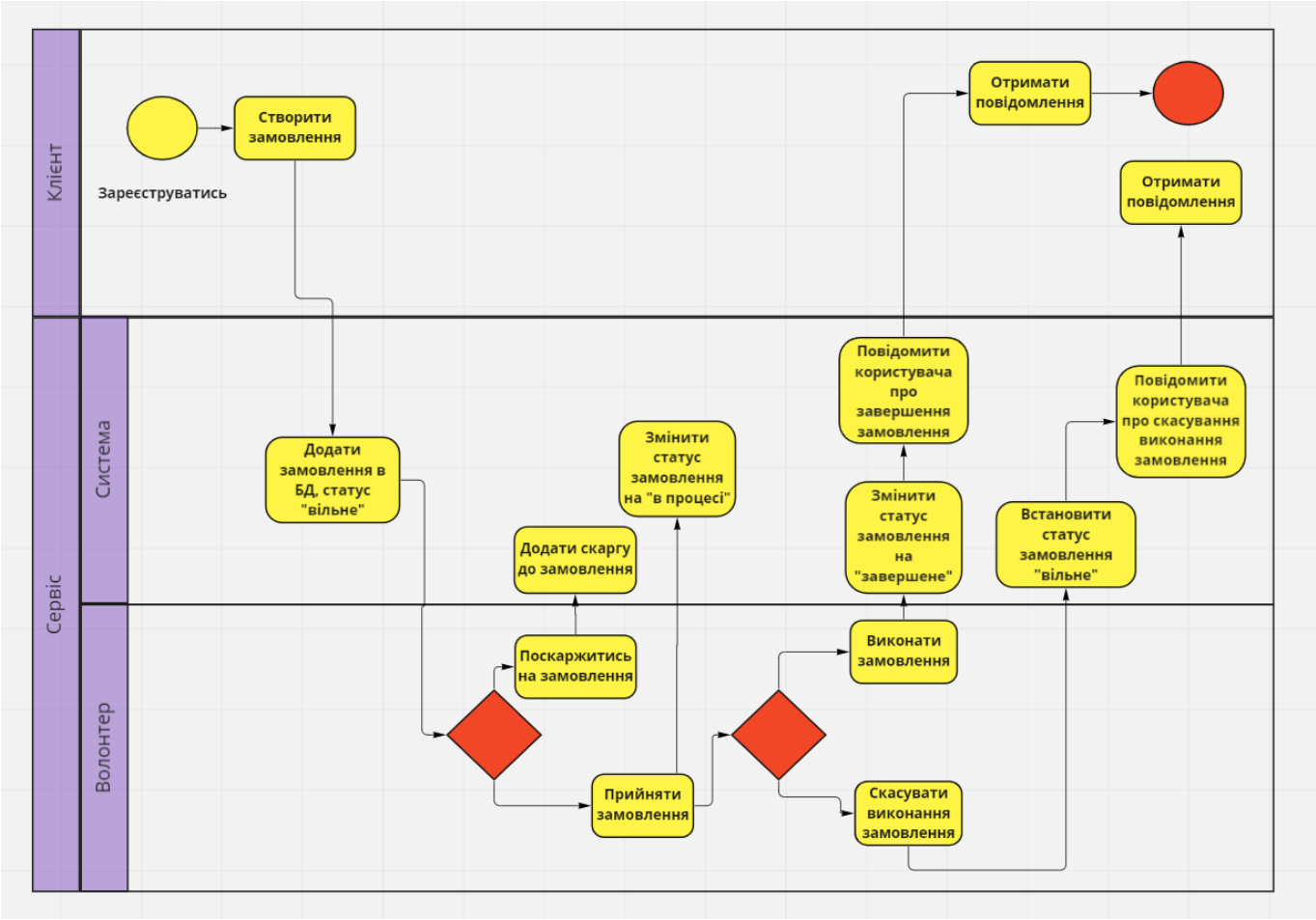

Рисунок 2.3 – Діаграма бізнес-процесів інтерактивного сервісу

Сервіс буде побудований на основі двох телеграм-ботів. Перший буде створений для користувачів і надаватиме користувачам можливість реєструватись створювати замовлення, переглядати власні дані, переглядати замовлення і дані про них та скасовувати замовлення. Також даний бот буде надсилати користувачу повідомлення у випадку повного виконання і подальшого завершення його замовлення та у випадку відмови волонтера від виконання замовлення. Другий телеграм-бот буде створений для

волонтерів. Даний бот буде з обмеженим доступом – для отримання доступу потрібно знати пароль, який видає організація. У боті для волонтерів буде можливість реєструватись, переглядати замовлення, приймати їх до виконання, або скаржитись на некоректне замовлення. Буде можливість переглядати замовлення в процесі виконання та завершені цим волонтером замовлення. Також буде наявний функціонал завершення замовлення, або скасування його виконання. Додатково цей бот надсилатиме повідомлення волонтеру, якщо користувач скасував замовлення, яке волонтер прийняв до виконання.

2.2.3 Вибір мови програмування

Telegram Bot API підтримує взаємодію з великою кількістю мов програмування, а саме Python, Node.js, Java, Rust, PHP.

1. Java – мова об’єктно-орєнтована. Написання коду за допомогою Java часто дуже детальне, а через це код буває громіздким.

Переваги Java:

- Об’єктно-орієнтована мова програмування;
- Портативність;
- Розповсюдженість;
- Багатопоточність.

Недоліки Java:

- Громіздкість коду;
- Низька швидкість;
- Платна ліцензія для комерційного використання.

2. Python – високорівнева мова програмування, орієнтована на продуктивність розробника, читабельність коду та його якість.

Переваги Python:

– Динамічна типізація. Хоча для когось динамічна типізація може здаватись мінусом, але для багатьох розробників, особливо з початковим досвідом, динамічна типізація зручніша. До того ж є можливість використовувати строгу типізацію;

– Універсальність у використанні;

– Простота синтаксису;

– Єдиний стандарт для написання коду.

Недоліки Python:

– Погана реалізація багатопоточності;

– Швидкість інтерпретації коду.

3. Node.js – оточення виконання, що базується на JavaScript.

Переваги Node.js:

– Оптимальна швидкість роботи додатків;

– Висока швидкість обробки запитів;

– Екосистема;

– Підтримка JSON.

Недоліки Node.js:

– Низька продуктивність при роботі з важкими задачами;

– Неідеальність через вік технології;

4. Rust – мультипагадигмальна мова програмування універсального призначення.

Переваги Rust:

– Лаконічний синтаксис;

– Аналізатор коду, що допомагає уникнути помилок при роботі з багатопоточністю.

– Надійна система взаємодії з пам’яттю;

– Доступний опис помилок.

Недоліки Rust:

– Немає типічних для ООП наслідування та класів, що робить код важким для сприйняття.

– Вимагає покриття 100% умов.

5. PHP – це скриптова мова, або «мова веб-розмітки», що найчастіше використовується для розробки веб-додатків.

Переваги PHP:

– Безкоштовне розповсюдження;

– Простота;

– Простота редагування;

– Велика спільнота користувачів.

Недоліки PHP:

– Низький рівень захисту;

– Застарілий;

Очевидно, обирати мову програмування потрібно базуючись на власних знаннях і зручності користування, однак варто також розуміти, що треба обирати підходящу мову програмування для виконання конкретних задач. Для створення телеграм-ботів для інтерактивної системи обробки замовлень обрано Python як найоптимальнішу мову програмування.

Висновки по розділу 2

Розроблено ER та фізичну моделі, що є основним процесом додатку. Обрана платформа для створення боту, а також створена діаграма бізнес-процесів і обрана мова програмування для розробки боту

РОЗДІЛ ІІІ. РЕАЛІЗАЦІЯ ІНТЕРАКТИВНОГО СЕРВІСУ ОБРОБКИ ЗАМОВЛЕНЬ

3.1 Створення бази даних

На відміну від більшості СУБД, при використанні SQLite БД створюється єдиним файлом, який інтегрується в програму, саме з цієї причини у SQLite відсутня система користувачів, а також в коді відсутнє «підключення» до БД. Через те, що БД створюється окремим файлом існує всього один спосіб створення БД.

- Відкрити SQLite;
- Обрати вкладку Databases;
- Створити новий файл БД.

3.1.1 Створення таблиць

Стандартний процес створення таблиць за допомогою мови SQL включає наступні дії:

1. Задати ім’я таблиці;
2. Перелічити усі стовпі із зазначенням типів даних атрибутів.
3. Перелічити обмеження для кожного стовпця:
   - первинний ключ;
   - зовнішній ключ (зв’язок з іншою таблицею даних);
   - допустимість невизначених значень;
   - обмеження на значення стовпчика;
   - задання значення за замовчуванням – значення, яке буде відображатись, якщо програмою не передбачено додавання специфічного значення;

При створенні таблиць з зовнішнім ключем, або зовнішніми ключами, потрібно враховувати те, що таблицю неможливо буде створити, якщо первинного ключа, на який посилається зовнішній ключ, не існує, тому важливо зберігати правильний порядок створення таблиць.

Таблиці створено згідно спроєктованої фізичну модель БД, а також дії, перелічені вище. Нижче наведений приклад створення однієї таблиці.

SQLite спрощує процес створення таблиць, оскільки дозволяє створити таблиці даних мануально, а також задавати обмеження за допомогою подвійного натискання на стовпець. Створення таблиці за допомогою вбудованого функціоналу СУБД SQLite (рис. 3.1).

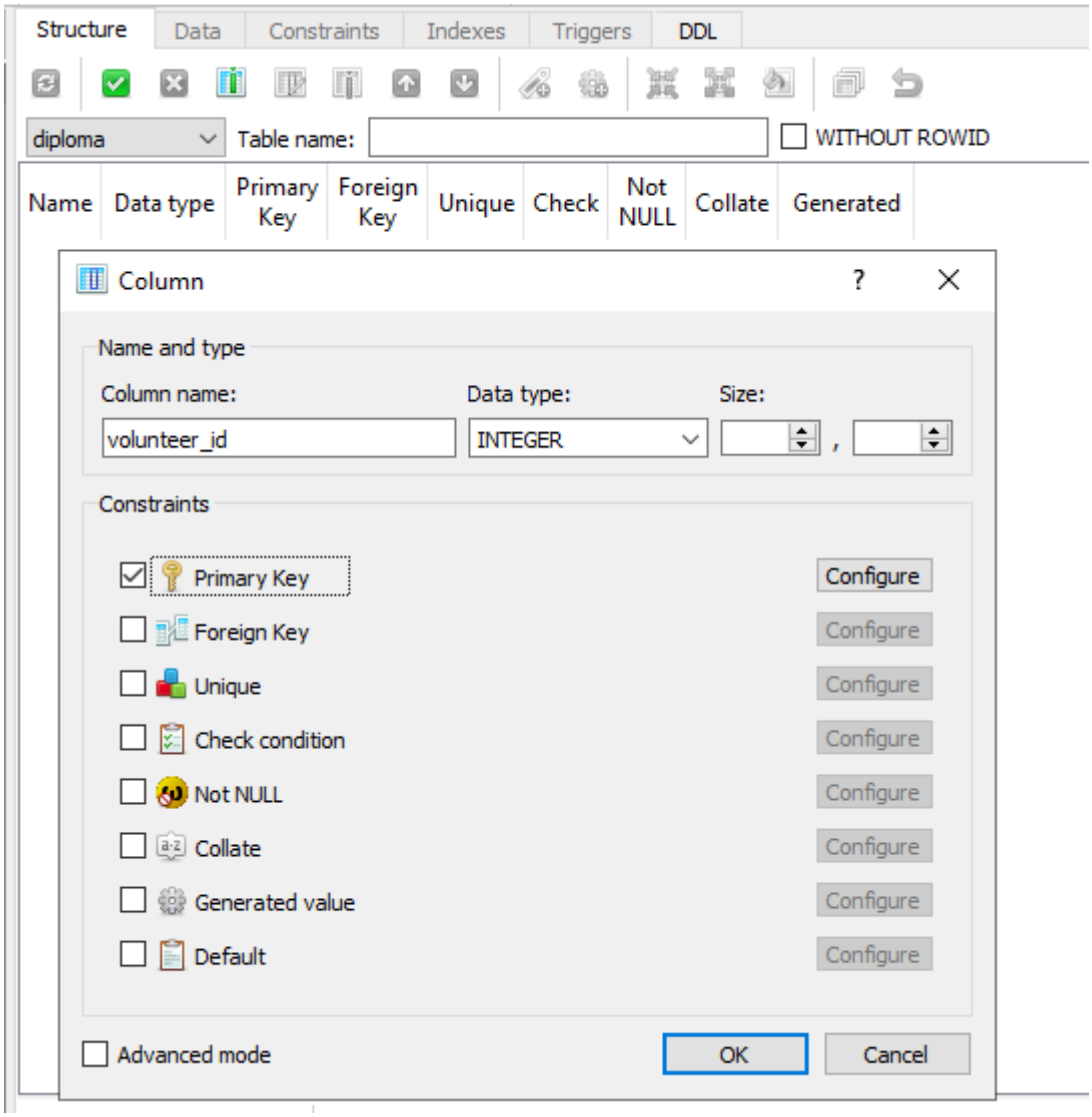

Рисунок 3.1 – Створення таблиці за допомогою вбудованого функціоналу СУБД SQLite

Також наявна можливість створити таблиці за допомогою SQL Editor. Приклад коду для створення таблиці:

```
CREATE TABLE volunteers (
volunteer_id INTEGER PRIMARY KEY AUTOINCREMENT
NOT NULL,
tg_volunteer_id INTEGER UNIQUE ON CONFLICT IGNORE
NOT NULL,
firstname STRING,
lastname STRING,
birth_date STRING
);
```

У SQLite синтаксис коду відрізняється від звичного SQL, тому нема можливості просто імпортувати SQL файл замість того, щоб писати скрипт. Після створення таблиць слід з'єднати їх, задавши зв'язок ключами.

3.2 Розробка програмного додатка

Створення телеграм-ботів на Python починається, як і будь який проєкт, з підключення бібліотек. Розглянемо бібліотеки, що будуть використовуватись для написання коду обох ботів, а саме:

- urllib.parse – відповідає за шифрування тексту для відправки по http/https.
- Requests – відповідає за виконання ботом http запитів. Дана бібліотека надає можливість використовувати так званий Web-hook – функція, для відправки повідомлення з бота до користувача, як в цьому ж самому боті, так і в іншому;
- Telebot – тобто Telegram Bot API. Бібліотека для створення боту. Для даного сервісу буде доцільним використати компонент types – користувацькі клавіатури, що являють собою кнопки з підв'язаними до них командами.

- Sqlite3 – власне, зовнішній модуль (бібліотека), що дозволяє підключитись до БД, створену за допомогою SQLite та використовувати її.
- Datetime – бібліотека для перевірки дати.

Для написання коду боту для волонтерів доведеться додати бібліотеку random – для показу вільних замовлень в довільному порядку.

Для коректної роботи реєстрації та створення замовлень, потрібно створити класи для даних користувачів та замовлень.

Бот користувачів:

```
class Order:
    def __init__(self, desc):
        self.desc = desc
        self.urgency = None
        self.region = None


class User:
    def __init__(self, name):
        self.first_name = name
        self.last_name = None
        self.sex = None
```

Бот волонтерів:

```
class Vol:
    def __init__(self, name):
        self.first_name = name
        self.last_name = None
        self.sex = None


class Order:
    def __init__(self, desc):
```

self.desc = desc

self.urgency = None

self.region = None

Також потрібно додати словники, в які будуть записуватись дані, що вводять користувачі, щоб зберігати ті дані в процесі виконання задач, а не передавати їх змінними в функціїї, що можна вважати наступними «кроками» боту.

user_dict = {}, order_dict = {} - бот для користувачів;

vol_dict = {}, order_delete_dict = {} – бот для волонтерів.

Далі у файлах ботів описані функції, що прив’язані до команд боту, таких, як /start, /register, тощо, та продовження тих функцій, оскільки бот функціонує покроково. Для прикладу, розглянемо частину коду з виконанням функції:

def add_user_to_table(chat_id): - функція додавання користувача до таблиці БД по завершенню реєстрації

try:

user = user_dict[chat_id] – відкриваємо словник за chat_id (унікальне значення для кожного користувача)

firstname = user.first_name – записано в змінну зі словника ім’я

lastname = user.last_name - записано в змінну зі словника прізвище

age = user.age – записано в змінну зі словника дату народження

bot.send_message(chat_id, "Ім'я: " + firstname + '\nПрізвище: ' + lastname + '\nДата народження: ' + age) – відправляє користувачу повідомлення з даними, що він ввів, які тільки що були записані в змінні.

db_users_insert(tg_user_id=chat_id, firstname=firstname, lastname=lastname, age=age) – завершення першого кроку і виклик функції запису даних користувача у БД

except Exception as e:
print(e)
pass

def db_users_insert(tg_user_id: int, firstname: str, lastname: str, age: str): присвоює змінним тип даних

cursor.execute('INSERT INTO users (tg_user_id, firstname, lastname, birth_date) VALUES (?, ?, ?, ?)', (tg_user_id, firstname, lastname, age)) виконує SQL запит, при цьому використовані так звані «плейсхолдери» - знаки питання замість значень. Це дозволяє захиститись від SQL-ін’єкцій, що підвищує степінь захисту БД. Далі відбувається передача в ці плейсхолдери значення змінних, тип даних яких було визначено на початку, тому в них SQL запит також не додати.

conn.commit() – підтверджує та зберігає зміни в БД.

Приблизно за таким самим алгоритмом виконуються майже всі функції бота.

Додатково, розглянемо різні перевірки.

1. Перевірка, чи не введена користувачем команда бота.

cmd_list={'/start': send_welcome, '/help': send_welcome, '/register': send_register, \

'/order': start_order, '/my_orders': my_orders, '/account': send_account, \

'/delete_order': delete_order} – був створений словник, в який додано всі команди бота у текстовому вигляді, та їхнім значенням встановлено функції, що викликають ці команди в боті. Далі була встановлена, для кожного введеного користувачем тексту, перевірка:

command = check_command(message)

if command:

command(message)

return

Перевірка спрацьовує в кожному кроці, після того, як користувач відправляє текст. Якщо текст, введений користувачем, співпадає з текстом зі словнику cmd_list, викликається функція, що присвоєна як значення тексту цій команді у словнику.

2. Перевірка формату віку користувача, реальності дати, року народження.

В процесі реєстрації користувач отримує запит від боту на введення дати народження. Формат, заданий користувачеві для дати є dd/mm/yyyy.

try:

datetime.datetime.strptime(age, "%d/%m/%Y") – в даному рядці коду завдяки бібліотеці datetime відбувається перевірка формату (яким символом розділяються день, місяць та рік), а також перевірка реальності дати. Дана бібліотека порівнює дані, введені користувачем, з реальними календарними даними в світі, отже у користувача відсутня можливість вказати 31-ий день листопада, оскільки в листопаді всього 30 днів, або 13-ий місяць.

day, month, year = age.split("/") – розділяє дані, введені користувачем, прибирає роздільний символ «/» та записує день, місяць та рік в різні змінні

if int(year) > 2006: - порівнює дані, що введені в рік зі значенням введеним користувачем

bot.send_message(chat_id, 'Ви занадто молоді для використання бота') – якщо рік більший, ніж 2006 – користувач отримує повідомлення, про недостатній вік для використання боту.

Return

if int(year) < 1932: - порівнює дані, що введені в рік зі значенням введеним користувачем

bot.send_message(chat_id, 'Ви, певно, вводите нереальний рік народження) – якщо рік менший, ніж 1932 – користувач отримує повідомлення, про нереальність введенного року народження.

return

except: - якщо рік потрапляє в діапазон підходящих, для віку користувача, але при цьому користувач вводить неіснуючий день, або місяць, то спрацьовує дана «помилка»

bot.send_message(chat_id, 'Введіть дату в коректному форматі') – бот надсилає користувачеві повідомлення «Введіть дату в коректному форматі»

bot.register_next_step_handler(message, process_age_step) - бот переводить користувача на повторне введення дати народження

return

bot.reply_to(message, "Ви успішно зареєстровані!") – якщо всі перевірки пройдені, то бот відпрявляє повідомлення користувачеві про успішну реєстрацію

add_user_to_table(chat_id) – виклик функції внесення даних користувача до БД.

Повний код телеграм-ботів представлено у додатку А даної роботи.

3.3 Взаємодія користувача з сервісом

Користувач має обмежені можливості, а саме – покрокова реєстрація, створення замовлення, перегляд даних, перегляд замовлень, скасування замовлення. Всі процеси, що являють собою додавання або видалення даних є покроковими. На рисунку 3.2 розглянуто процес покрокового створення замовлення користувача.

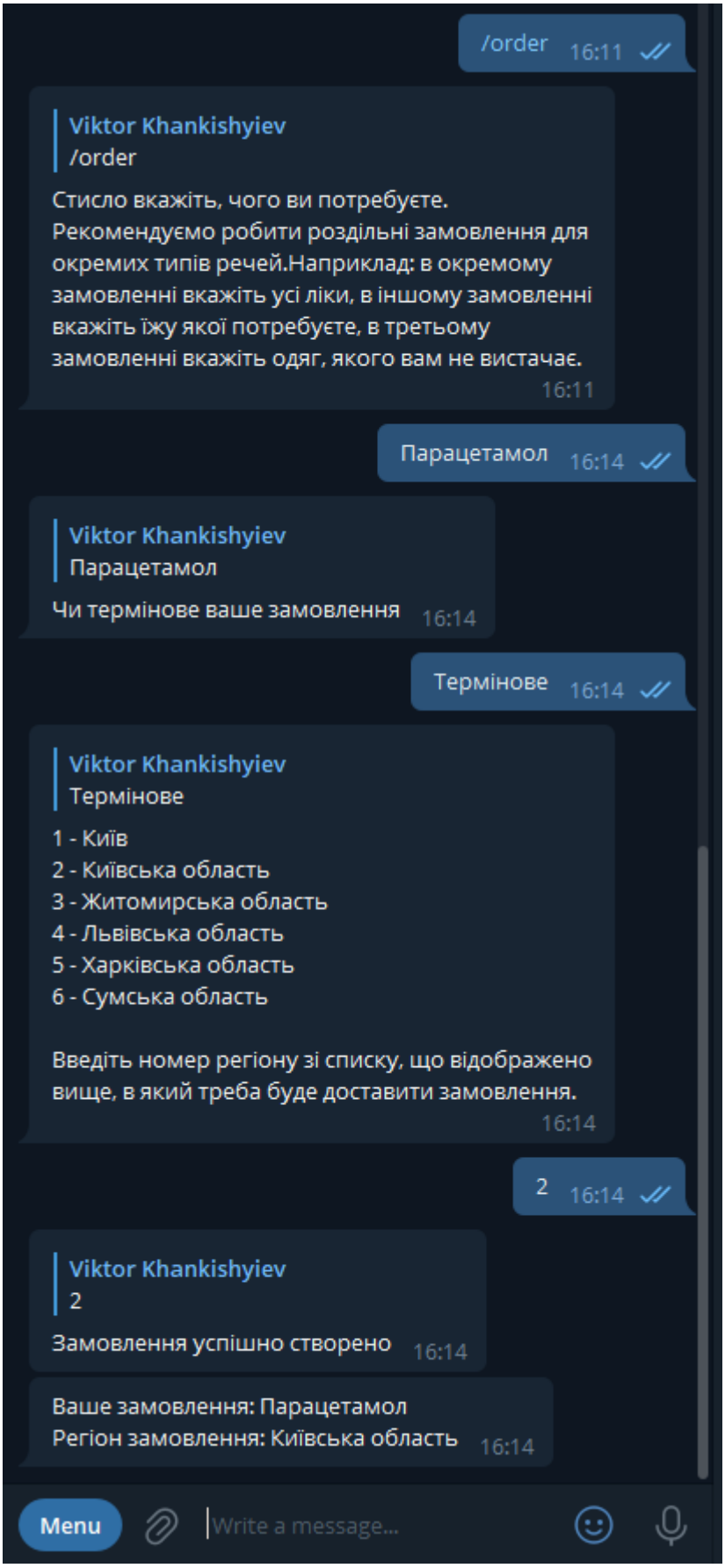

Рисунок 3.2 – створення замовлення

На рисунку 3.3 представлено перегляд персональних даних користувача.

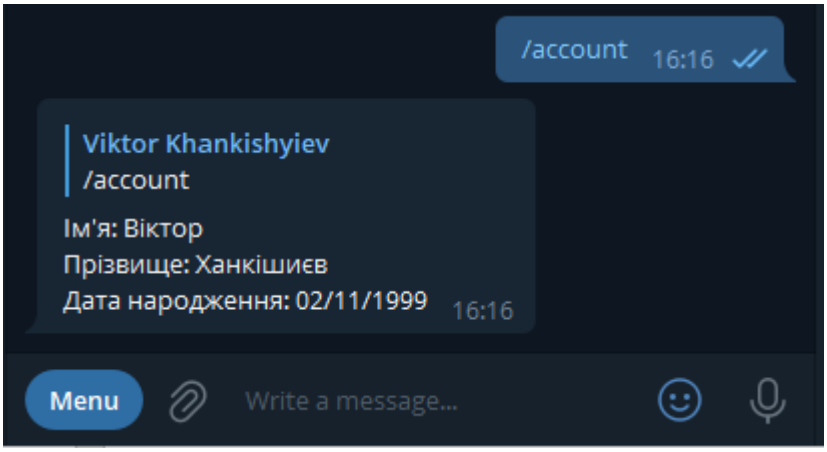

Рисунок 3.3 – перегляд даних користувача

На рисунку 3.4 представлено процес скасування замовлення користувачем.

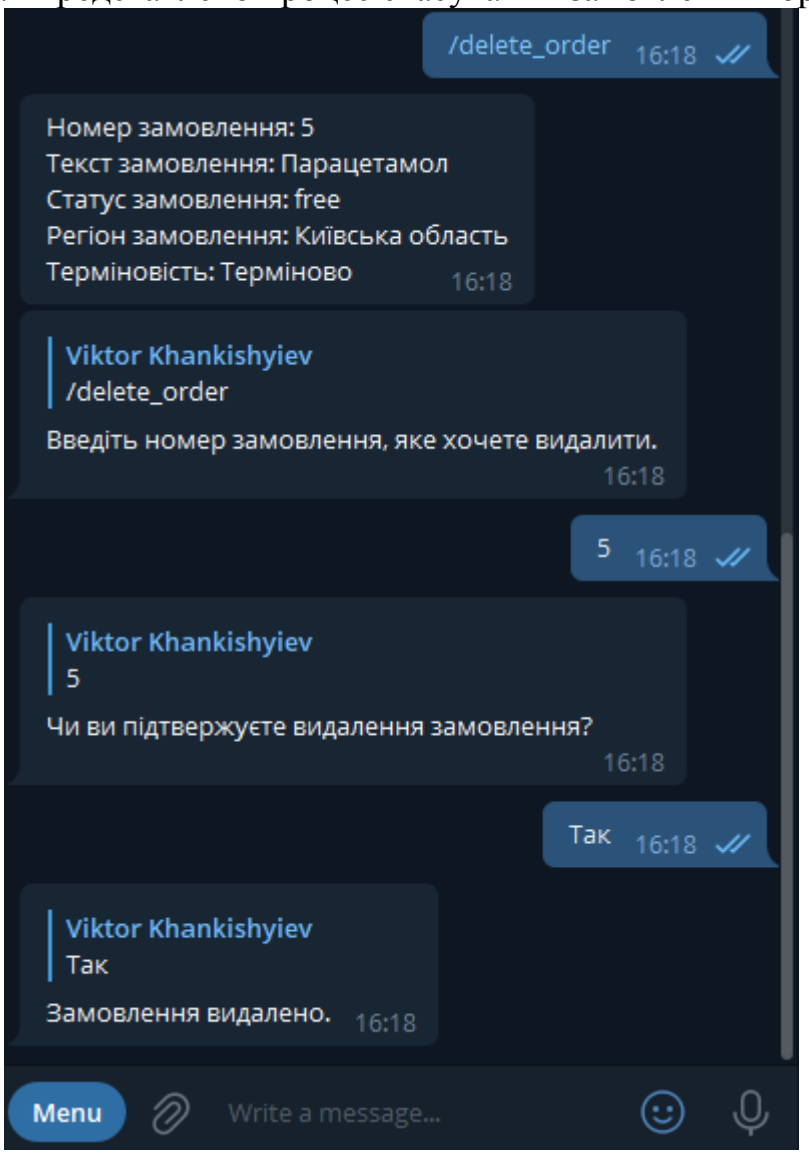

Рисунок 3.4 – скасування замовлення користувачем

Висновки по розділу 3

Було створено БД, та розроблено телеграм ботів, розглянуто біблотеки, що використовуються у функціоналі бота. Додатково розглянуто класи додатку, користувацькі словники та перевірки для коректності роботи бота.

ВИСНОВОК

Проаналізувавши предметну область, структуру та функціонал інтерактивного сервісу обробки замовлень, зроблений висновок, що попри існуюче різноманіття доступних технологій та рішень для даних задач, питання обробки замовлень все ще не доведене та не скоро буде доведене до ідеалу. А отже, прийнято рішення спробувати вирішити це питання за допомогою створення нового сервісу, простого та інтуїтивного у використанні, який покликаний зменшити витрати на персонал, оренду та розробку у невеликих організацій (комерційних та громадських) та благодійних фондів, а також покликаний спростити процес створення замовлення для пересічного користувача.

Створено модель бізнес-процесів, а отже визначений функціонал та порядок дій користувача та волонтера для коректної роботи системи.

Створені ER та фізична моделі, визначені основні сутності та побудовані зв’язки між ними, спроектовано реляційну БД для сервісу, визначено первинний та зовнішній ключ для кожної таблиці бази даних.

Створення бази даних виконано на основі СУБД SQLite, за допомогою Python та Telegram Bot API створені боти, для використання користувачами та волонтерами, а також створені перевірки всередині цих ботів, ціль яких полягає в тому, щоб не допустити збої та некоректну роботу процесів всередині системи.

Дана розробка є базовою, тобто містить лише базові функції, але може бути використана для організації обробки більш складних процесів обробки замовлень і може бути підлаштована під потреби твєї організації, яка збирається її використовувати.

ПЕРЕЛІК ПОСИЛАНЬ

1. Telegram Bot API Documentation. Електронний ресурс – https://core.telegram.org/api

2. Paul Sanderson. SQLite Forensics. 2018

3. Опитування українців про основний месенджер. Електронний ресурс – https://ain.ua/ru/2021/01/26/dlya-506-telegram-osnovnoj-messendzher-rezultaty-oprosa-ain-ua/

4. What is an Entity Relationship Diagram (ERD)? Електронний ресурс - https://www.lucidchart.com/pages/er-diagrams

5. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Martin Fowler. 2003

6. SQL QuickStart Guide. Walter Shields. 2019

7. SQL Cookbook. Anthony Molinaro. 2006

8. Database Systems: The Complete Book. Hector Garcia-Molina, Jeff Ullman, and Jennifer Widom..2001

9. Изучаем SQL. Генерация, выборка и обработка данных. 3-е издание - Алан Болье. 2021

10. Python. Самое необходимое. Н. А. Прохоренок. 2011

11. Python 3.10.5 documentation. Електронний ресурс – https://docs.python.org/3/

12. Введение в UML от создателей языка. Гради Буч. 1998

13. Data Modeling Made Simple: A Practical Guide for Business and IT Professionals, 2nd Edition. Steve Hoberman. 2015

14. Классификация СУБД. Типы и виды СУБД. Електронний ресурс – https://www.sqlhome.org.ua/klassifikaciya-tipy-vidy-subd/

15. Що таке ER Modeling? Дізнайтеся на прикладі. Електронний ресурс – https://uk.csstricks.net/8224980-what-is-er-modeling-learn-with-example

16. Карпова И.П. Базы данных. Учебное пособие. – М., 2009

17. Что такое NoSQL? Електронний ресурс – https://aws.amazon.com/ru/nosql/

18. Python Cookbook, Third Edition. David Beazley, Brian Jones. 2013

ДОДАТОК А

```
import urllib.parse

import requests

import telebot

from telebot import types

import sqlite3

import datetime


API_TOKEN = '5315016907:AAHzS9bOjvaJ6tGxonTCMH-i_fTaC98sBEc'

bot = telebot.TeleBot(API_TOKEN)


conn = sqlite3.connect('diploma.db', check_same_thread=False)

cursor = conn.cursor()



userInputData = {}


user_dict = {}


order_dict = {}


order_delete_dict = {}
```

```
class Order:
    def __init__(self, desc):
        self.desc = desc
        self.urgency = None
        self.region = None


class User:
    def __init__(self, name):
        self.first_name = name
        self.last_name = None
        self.sex = None


# Handle '/start' and '/help'
@bot.message_handler(commands=['help', 'start'])
def send_welcome(message):
    chat_id = message.chat.id
    # try:
    #     u = user_dict[chat_id]
    #     bot.reply_to(message, "Вже зареєстрований")
    # except KeyError:
     msg = bot.reply_to(message, "За командою /register ви можете зареєструватися.\nЗа командою /account ви можете переглянути ваші дані.\n\
```

За командою /order ви можете створити замовлення.\nЗа командою /my_orders ви можете переглянути свої замовлення.\n\

За командою /delete_order ви можете видалити ваше замовлення.", reply_markup=types.ReplyKeyboardRemove())

```
# /register
@bot.message_handler(commands=['register'])
def send_register(message):
    sql=f'SELECT id FROM users WHERE tg_user_id = {message.chat.id}'
    cursor.execute(sql)
    check_user_availability=cursor.fetchall()
    # check_user_availability=cursor.fetchall()[0][0]
    if check_user_availability:
                    bot.send_message(message.chat.id, 'Ви вже зареєстровані!', reply_markup=types.ReplyKeyboardRemove())
        return
    msg = bot.reply_to(message, "Введіть своє ім'я (без прізвища)")
    bot.register_next_step_handler(msg, process_firstname_step)

def process_firstname_step(message):
    try:
        command = check_command(message)
        if command:
```

```
            command(message)
            return
        chat_id = message.chat.id
        first_name = message.text
        user = User(first_name)
        user_dict[chat_id] = user
        msg = bot.reply_to(message, "Введіть своє прізвище")
        bot.register_next_step_handler(msg, process_lastname_step)
    except Exception as e:
        bot.reply_to(message, "Ви не ввели ім'я")


def process_lastname_step(message):
    try:
        command = check_command(message)
        if command:
            command(message)
            return
        chat_id = message.chat.id
        last_name = message.text
        user = user_dict[chat_id]
        user.last_name = last_name
        msg = bot.reply_to(message, "Введіть дату народження (у форматі dd/mm/yyyy)")
```

```
        bot.register_next_step_handler(msg, process_age_step)
    except Exception as e:
        bot.reply_to(message, "Ви не ввели прізвище")


def process_age_step(message):
    try:
        command = check_command(message)
        if command:
            command(message)
            return
        chat_id = message.chat.id
        age = message.text
        user = user_dict[chat_id]
        user.age = age
        print(message.text)
        print(type(message.text))
        try:
            date_of_birth = datetime.datetime.strptime(age, "%d/%m/%Y")
            day, month, year = age.split("/")
            print(day)
            print(month)
            print(year)
```

```
            if int(year) > 2006:
                bot.send_message(chat_id, 'Ви занадто молоді для використання бота')
                return
        except:
            bot.send_message(chat_id, 'Введіть дату в коректному форматі')
            bot.register_next_step_handler(message, process_age_step)
            return
        bot.reply_to(message, "Ви успішно зареєстровані!")
        add_user_to_table(chat_id)
    except Exception as e:
        bot.reply_to(message, 'Ви не ввели ваш вік')


def add_user_to_table(chat_id):
    try:
        user = user_dict[chat_id]
        firstname = user.first_name
        lastname = user.last_name
        age = user.age
        # tg_user_id=user.chat_id
        #print(user_dict)
         bot.send_message(chat_id, "Ім'я: " + firstname + '\n Прізвище: ' + lastname + '\n
Дата народження: ' + age)
```

```
            db_users_insert(tg_user_id=chat_id, firstname=firstname, lastname=lastname, age=age)
    except Exception as e:
        print(e)
        pass


def db_users_insert(tg_user_id: int, firstname: str, lastname: str, age: str):
     cursor.execute('INSERT INTO users (tg_user_id, firstname, lastname, birth_date) VALUES (?, ?, ?, ?)', (tg_user_id, firstname, lastname, age))
    conn.commit()


# /account
@bot.message_handler(commands=['account'])
def send_account(message):
    sql=f'SELECT id FROM users WHERE tg_user_id = {message.chat.id}'
    cursor.execute(sql)
    check_user_availability=cursor.fetchall()
    # check_user_availability=cursor.fetchall()[0][0]
    if not check_user_availability:
        bot.send_message(message.chat.id, 'Ви ще не зареєстровані!')
        send_welcome(message)
        return
```

```
    sql = f"SELECT firstname, lastname, birth_date FROM users WHERE tg_user_id = {message.chat.id}"
    cursor.execute(sql)
    user = cursor.fetchall()
        reply = f"Ім'я: {user[0][0]} \nПрізвище: {user[0][1]} \nДата народження: {user[0][2]}"
    msg = bot.reply_to(message, reply, reply_markup=types.ReplyKeyboardRemove())


# /order
@bot.message_handler(commands=['order'])
def start_order(message):
    sql=f'SELECT id FROM users WHERE tg_user_id = {message.chat.id}'
    cursor.execute(sql)
    check_user_availability=cursor.fetchall()
    # check_user_availability=cursor.fetchall()[0][0]
    if not check_user_availability:
        bot.send_message(message.chat.id, 'Ви ще не зареєстровані!')
        send_welcome(message)
        return
    cursor.execute(f'SELECT reports FROM users WHERE tg_user_id={message.chat.id}')
    reports=cursor.fetchall()[0][0]
    if reports>9:
```

bot.send_message(message.chat.id, 'Через велику кількість скарг на ваші замовлення ви не маєте доступу до створення нових замовлень')

send_welcome(message)

return

msg = bot.reply_to(message, "Стисло вкажіть, чого ви потребуєте. Рекомендуємо робити роздільні замовлення для окремих типів речей.\

Наприклад: в окремому замовленні вкажіть усі ліки, в іншому замовленні вкажіть їжу якої потребуєте, \

в третьому замовленні вкажіть одяг, якого вам не вистачає.", reply_markup=types.ReplyKeyboardRemove())

bot.register_next_step_handler(msg, process_urgency_order_step)

def process_urgency_order_step(message):

try:

command = check_command(message)

if command:

command(message)

return

chat_id = message.chat.id

order_dict[chat_id] = Order(message.text)

markup = types.ReplyKeyboardMarkup(one_time_keyboard=True)

markup.add('Термінове', 'Нетермінове')

msg = bot.reply_to(message, 'Чи термінове ваше замовлення', reply_markup=markup)

```
        bot.register_next_step_handler(msg, process_next_order_step)
    except Exception as e:
        bot.reply_to(message, 'oooops')


def process_next_order_step(message):
    try:
        command = check_command(message)
        if command:
            command(message)
            return
        chat_id = message.chat.id
        order_urg = message.text
        order = order_dict[chat_id]
        if (order_urg == 'Термінове'):
            order.urgency = 1
        elif (order_urg == 'Нетермінове'):
            order.urgency = 2
        else:
            msg=bot.send_message(message,'Оберіть степінь терміновості замовлення')
            bot.register_next_step_handler(msg, process_next_order_step)
            return
        sql=f'SELECT * FROM regions'
```

```
        cursor.execute(sql)
        regions_ids=cursor.fetchall()
        regions_ids_str = ''
        for i in regions_ids:
            regions_ids_str += f'{i[0]} - {i[1]}\n'
        msg= bot.reply_to(message, f'{regions_ids_str} \nВведіть номер регіону зі списку, що відображено вище, в який треба буде доставити замовлення.', reply_markup=types.ReplyKeyboardRemove())
        bot.register_next_step_handler(msg, process_order_region_step)
    except Exception as e:
        bot.reply_to(message, 'oooops')


def process_order_region_step(message):
    try:
        command = check_command(message)
        if command:
            command(message)
            return
        chat_id = message.chat.id
        sql=f'SELECT region_id FROM regions'
        cursor.execute(sql)
        regions_ids=cursor.fetchall()
        ids = [i[0] for i in regions_ids]
```

```
        if not int(message.text) in ids:
            raise Exception
        order_region = int(message.text)
        order = order_dict[chat_id]
        order.region = order_region
        bot.reply_to(message, 'Замовлення успішно створено')
        add_order_to_table(message)
    except Exception as e:
        bot.reply_to(message, 'you oooops')


def add_order_to_table(message):
    try:
        chat_id = message.chat.id
        order = order_dict[chat_id]
        order_desc = order.desc
        order_urgency_id = order.urgency
        region_id=order.region
        sql2=f'SELECT region_description FROM regions WHERE region_id = {region_id}'
        cursor.execute(sql2)
        region_desc=cursor.fetchall()[0][0]
        sql1=f'SELECT id FROM users WHERE tg_user_id = {message.chat.id}'
        cursor.execute(sql1)
```

```
        user_id=cursor.fetchall()[0][0]
        bot.send_message(chat_id, f'Ваше замовлення: {order_desc} \nРегіон замовлення: {region_desc}')
            db_orders_insert(chat_id = chat_id, user_id=user_id, order_desc=order_desc, order_urgency_id=order_urgency_id, region_id=region_id)
    except Exception as e:
        print(e)
        pass


def db_orders_insert(chat_id: int, user_id: int, order_desc: str, order_urgency_id: int, region_id: int):
    try:
            cursor.execute('INSERT INTO orders (order_desc, order_status, user_id, volunteer_id, order_urgency_id, region_id) VALUES (?, ?, ?, ?, ?, ?)', (order_desc, 'free', user_id, None, order_urgency_id, region_id))
        conn.commit()
        order_dict.pop(chat_id)
    except Exception as e:
        print(e)


# /my_orders
@bot.message_handler(commands=['my_orders'])
def my_orders(message):
    sql=f'SELECT id FROM users WHERE tg_user_id = {message.chat.id}'
```

```
    cursor.execute(sql)
    check_user_availability=cursor.fetchall()
    # check_user_availability=cursor.fetchall()[0][0]
    if not check_user_availability:
        bot.send_message(message.chat.id, 'Ви ще не зареєстровані!')
        send_welcome(message)
        return
        select_orders=f'SELECT orders.order_id, orders.order_desc, orders.order_status, regions.region_description, order_urgency.order_urgency_desc \
    FROM orders join users on orders.user_id = users.id join regions on orders.region_id = regions.region_id \
      join order_urgency on orders.order_urgency_id = order_urgency.order_urgency_id WHERE users.tg_user_id = {message.chat.id}'
    cursor.execute(select_orders)
    user_orders=cursor.fetchall()
    for order in user_orders:
            bot.send_message(message.chat.id, f'Номер замовлення: {order[0]}\nТекст замовлення: {order[1]}\nСтатус замовлення: {order[2]}\nРегіон замовлення: {order[3]}\nТерміновість: {order[4]}')


# /delete_order
@bot.message_handler(commands=['delete_order'])
def delete_order(message):
    sql=f'SELECT id FROM users WHERE tg_user_id = {message.chat.id}'
```

```
    cursor.execute(sql)
    check_user_availability=cursor.fetchall()
    # check_user_availability=cursor.fetchall()[0][0]
    if not check_user_availability:
        bot.send_message(message.chat.id, 'Ви ще не зареєстровані!')
        send_welcome(message)
        return
        select_orders=f'SELECT orders.order_id, orders.order_desc, orders.order_status, regions.region_description, order_urgency.order_urgency_desc \
    FROM orders join users on orders.user_id = users.id join regions on orders.region_id = regions.region_id \
    join order_urgency on orders.order_urgency_id = order_urgency.order_urgency_id \
    WHERE users.tg_user_id = {message.chat.id}'
    cursor.execute(select_orders)
    user_orders=cursor.fetchall()
    for order in user_orders:
            bot.send_message(message.chat.id, f'Номер замовлення: {order[0]}\nТекст замовлення: {order[1]}\nСтатус замовлення: {order[2]}\nРегіон замовлення: {order[3]}\nТерміновість: {order[4]}')
    msg = bot.reply_to(message, "Введіть номер замовлення, яке хочете видалити.")
    bot.register_next_step_handler(msg, process_delete_order_step)


def process_delete_order_step(message):
```

```
    command = check_command(message)
    if command:
        command(message)
        return
        select_orders=f'SELECT orders.order_id, orders.order_desc, orders.order_status, regions.region_description, order_urgency.order_urgency_desc \
    FROM orders join users on orders.user_id = users.id join regions on orders.region_id = regions.region_id \
     join order_urgency on orders.order_urgency_id = order_urgency.order_urgency_id WHERE users.tg_user_id = {message.chat.id}'
    cursor.execute(select_orders)
    user_orders=cursor.fetchall()
    ids = [i[0] for i in user_orders]
    if message.text.isdigit() and int(message.text) in ids:
        order_delete_dict[message.chat.id] = int(message.text)
        markup = types.ReplyKeyboardMarkup(one_time_keyboard=True)
        markup.add('Так', 'Ні')
            msg = bot.reply_to(message, 'Чи ви підтвержуєте видалення замовлення?', reply_markup=markup)
        bot.register_next_step_handler(msg, process_confirm_delete_order_step)
    else:
        msg = bot.reply_to(message, "Введіть правильний номер замовлення.")
        bot.register_next_step_handler(msg, process_delete_order_step)
```

```
def process_confirm_delete_order_step(message):
    command = check_command(message)
    if command:
        command(message)
        return
    if message.text == 'Так':
        order_id = order_delete_dict[message.chat.id]
        select_volunteer=f'SELECT volunteers.tg_volunteer_id \
            FROM orders JOIN volunteers on orders.volunteer_id = volunteers.volunteer_id \
            WHERE orders.order_id = {order_id} AND orders.order_status = "progress"'
        cursor.execute(select_volunteer)
        volunteer=cursor.fetchall()
        if volunteer:
                                select_order=f'SELECT orders.order_id, orders.order_desc, regions.region_description \
                FROM orders JOIN regions on orders.region_id = regions.region_id WHERE orders.order_id = {order_id}'
            cursor.execute(select_order)
            order=cursor.fetchall()
             volunteer_message = f'Замовник видалив замовлення: \n\nНомер замовлення: {order[0][0]}\nТекст замовлення: {order[0][1]}\nРегіон замовлення: {order[0][2]}'
            volunteer_message_url = urllib.parse.quote(volunteer_message)
```

```
requests.get(f'https://api.telegram.org/bot5372774502:AAEg4cQlZGLjj8QFtwmu1NxXALEy89wbj08/sendMessage?chat_id={volunteer[0][0]}&text={volunteer_message_url}')
        delete_order_sql = f'DELETE FROM orders WHERE order_id = {order_id}'
        cursor.execute(delete_order_sql)
        conn.commit()
        order_delete_dict.pop(message.chat.id)
                                bot.reply_to(message, 'Замовлення видалено.', reply_markup=types.ReplyKeyboardRemove())

def check_command(message):
    try:
        command = cmd_list[message.text]
        return command
    except KeyError:
        return None

cmd_list={'/start': send_welcome, '/help': send_welcome, '/register': send_register, \
    '/order': start_order, '/my_orders': my_orders, '/account': send_account, \
    '/delete_order': delete_order}

 #Enable saving next step handlers to file "./.handlers-saves/step.save".
```

```
#Delay=2 means that after any change in next step handlers (e.g. calling register_next_step_handler())
#saving will hapen after delay 2 seconds.
bot.enable_save_next_step_handlers(delay=2)

#Load next_step_handlers from save file (default "./.handlers-saves/step.save")
#WARNING It will work only if enable_save_next_step_handlers was called!
#bot.load_next_step_handlers()

if __name__ == '__main__':
    bot.polling(none_stop=True)

one_time_keyboard=True
from tracemalloc import stop
import telebot
from telebot import types
from telebot.types import InlineKeyboardMarkup, InlineKeyboardButton
import sqlite3
import random
import datetime
import urllib.parse
import requests
```

```
password= "Pass word"

API_TOKEN = '5372774502:AAEg4cQlZGLjj8QFtwmu1NxXALEy89wbj08'
bot = telebot.TeleBot(API_TOKEN)

conn = sqlite3.connect('diploma.db', check_same_thread=False)
cursor = conn.cursor()


userInputData = {}

vol_dict = {}

order_delete_dict = {}

class Vol:
    def __init__(self, name):
        self.first_name = name
        self.last_name = None
        self.sex = None

class Order:
    def __init__(self, desc):
```

```
        self.desc = desc
        self.urgency = None
        self.region = None

def gen_order_markup():
    markup = InlineKeyboardMarkup()
    markup.row_width = 2
    markup.add(InlineKeyboardButton("Взяти", callback_data="order_accept"),
               InlineKeyboardButton("Далі", callback_data="order_next"),
               InlineKeyboardButton("Скарга", callback_data="order_report"),
               InlineKeyboardButton("В меню", callback_data="order_menu"))
    return markup

@bot.callback_query_handler(func=lambda call: True)
def callback_query(call):
    if call.data == "order_accept":

        bot.answer_callback_query(call.id, "order_accept")
    elif call.data == "order_next":
        bot.answer_callback_query(call.id, "order_next")
    elif call.data == "order_report":
        bot.answer_callback_query(call.id, "order_report")
```

```
    elif call.data == "order_menu":
        bot.answer_callback_query(call.id, "order_menu")


# /help
@bot.message_handler(commands=['help'])
def process_help(message):
    chat_id=message.chat.id
    bot.send_message(chat_id, '/start - почати роботу з ботом\
\n/help - переглянути список команд\
\n/register - реєстрація\
\n/orders - переглянути замовлення\
\n/my_orders - ваші активні замовлення\
\n/finished_orders - ваші завершені замовлення\
\n/close_order - завершити замовлення\
\n/remove_order - відмовитись від замовлення', reply_markup=types.ReplyKeyboardRemove())


# /start
@bot.message_handler(commands=['start'])
def send_welcome(message):
    chat_id = message.chat.id
    sql = f"SELECT * FROM volunteers WHERE tg_volunteer_id = {chat_id}"
    cursor.execute(sql)
```

```
    volunteer = cursor.fetchall()
    if volunteer:
                        msg = bot.reply_to(message, "Вітаємо! /start", reply_markup=types.ReplyKeyboardRemove())
    else:
        msg = bot.reply_to(message, "Це бот для волонтерів. Для продовження введіть пароль:")
        bot.register_next_step_handler(msg, process_password_step)


def process_password_step(message):
    try:
        command = check_command(message)
        if command:
            command(message)
            return
        if message.text != password:
            msg = bot.reply_to(message, "Неправильний пароль. Спробуйте ще раз:")
            bot.register_next_step_handler(msg, process_password_step)
            return
        chat_id = message.chat.id
        print(chat_id)
            cursor.execute('INSERT INTO volunteers (tg_volunteer_id, firstname, lastname, birth_date) VALUES (?, ?, ?, ?)', (chat_id, None, None, None))
```

```
        conn.commit()
        msg = bot.reply_to(message, "Доступ до боту відкритий.")
        process_help(message)
        # process_register(message)
    except Exception as e:
        print(e)
        bot.reply_to(message, "Ви не ввели пароль")


# /register
@bot.message_handler(commands=['register'])
def process_register(message):
        sql=f'SELECT firstname FROM volunteers WHERE tg_volunteer_id = {message.chat.id}'
    cursor.execute(sql)
    check_user_availability=cursor.fetchall()
    if not check_user_availability:
        send_welcome(message)
        return
    if check_user_availability[0][0]:
                    bot.send_message(message.chat.id, 'Ви вже зареєстровані!', reply_markup=types.ReplyKeyboardRemove())
        return
    msg = bot.reply_to(message, "Введіть своє ім'я (без прізвища)")
```

```
    bot.register_next_step_handler(msg, process_firstname_step)


def process_firstname_step(message):
    try:
        command = check_command(message)
        if command:
            command(message)
            return
        chat_id = message.chat.id
        first_name = message.text
        user = Vol(first_name)
        vol_dict[chat_id] = user
        msg = bot.reply_to(message, "Введіть своє прізвище")
        bot.register_next_step_handler(msg, process_lastname_step)
    except Exception as e:
        bot.reply_to(message, "Ви не ввели ім'я")


def process_lastname_step(message):
    try:
        command = check_command(message)
        if command:
            command(message)
```

```
            return
        chat_id = message.chat.id
        last_name = message.text
        user = vol_dict[chat_id]
        user.last_name = last_name
        msg = bot.reply_to(message, "Введіть дату народження (у форматі dd/mm/yyyy)")
        bot.register_next_step_handler(msg, process_age_step)
    except Exception as e:
        bot.reply_to(message, "Ви не ввели прізвище")


def process_age_step(message):
    try:
        command = check_command(message)
        if command:
            command(message)
            return
        chat_id = message.chat.id
        age = message.text
        user = vol_dict[chat_id]
        user.age = age
        try:
            date_of_birth = datetime.datetime.strptime(age, "%d/%m/%Y")
```

```
            day, month, year = age.split("/")
            print(day)
            print(month)
            print(year)
            if int(year) > 2004:
                bot.send_message(chat_id, 'Ви занадто молоді для використання бота')
                return
        except:
            # print("Incorrect date!")
            bot.send_message(chat_id, 'Введіть дату в коректному форматі')
            bot.register_next_step_handler(message, process_age_step)
            return
        bot.reply_to(message, "Ви успішно зареєстровані!")
        add_user_to_table(chat_id)
    except Exception as e:
        bot.reply_to(message, 'Ви не ввели ваш вік')


def add_user_to_table(chat_id):
    try:
        user = vol_dict[chat_id]
        firstname = user.first_name
        lastname = user.last_name
```

```
        age = user.age
        bot.send_message(chat_id, "Ім'я: " + firstname + '\nПрізвище: ' + lastname + '\nДата народження: ' + age)
        db_users_insert(chat_id, firstname=firstname, lastname=lastname, age=age)
    except Exception as e:
        print(e)
        pass


def db_users_insert(chat_id, firstname: str, lastname: str, age: str):
    cursor.execute(f'UPDATE volunteers SET firstname=?, lastname=?, birth_date=? WHERE tg_volunteer_id={chat_id}' , (firstname, lastname, age))
    conn.commit()


# /orders
@bot.message_handler(commands=['orders'])
def send_order(message, id=None):
        sql=f'SELECT firstname FROM volunteers WHERE tg_volunteer_id = {message.chat.id}'
    cursor.execute(sql)
    check_user_availability=cursor.fetchall()
    print(check_user_availability)
    if not check_user_availability:
        send_welcome(message)
```

```
        return
    if check_user_availability[0][0] is None:
        bot.send_message(message.chat.id, 'Ви не зареєстровані!')
        send_welcome(message)
        return
    try:
                sql1 = f"SELECT DISTINCT orders.order_id, orders.order_desc, order_urgency.order_urgency_desc, regions.region_description, reports.volunteer_id \
                    FROM orders JOIN order_urgency on orders.order_urgency_id = order_urgency.order_urgency_id \
            JOIN regions on orders.region_id = regions.region_id LEFT JOIN reports on orders.order_id = reports.order_id \
        LEFT JOIN volunteers on reports.volunteer_id = volunteers.volunteer_id \
                    WHERE (volunteers.tg_volunteer_id != {message.chat.id} OR volunteers.tg_volunteer_id IS NULL) AND (orders.order_status = 'free')"
        cursor.execute(sql1)
        orders = cursor.fetchall()
        print(orders)
        l = len(orders)
        i = random.randint(0, (l-1))
        if l > 1:
            while orders[i][0] == id:
                i = random.randint(0, (l-1))
```

```
        markup = types.ReplyKeyboardMarkup(one_time_keyboard=True)
        markup.add('Взяти', 'Далі', 'Скарга', 'В меню')
            msg = bot.reply_to(message, f'Номер замовлення: {orders[i][0]}\nТекст замовлення: {orders[i][1]}\nСтепінь терміновості: {orders[i][2]}\nРегіон замовлення: {orders[i][3]}', reply_markup=markup)
        bot.register_next_step_handler(msg, process_send_order, id=orders[i][0])
    except Exception as e:
        bot.reply_to(message, 'Вільні замовлення відсутні')
        process_help(message)


def process_send_order(message, id):
    try:
        command = check_command(message)
        if command:
            command(message)
            return
        chat_id=message.chat.id
        reply = message.text
        if reply == 'Взяти':
            bot.send_message (chat_id, 'Замовлення взяте')
                sql_volunteer_id=f'SELECT volunteer_id FROM volunteers WHERE tg_volunteer_id = {message.chat.id}'
            cursor.execute(sql_volunteer_id)
```

```
            volunteer_id=cursor.fetchall()[0][0]
            process_take_order(id=id, volunteer_id=volunteer_id)
        elif reply == 'Далі':
            send_order(message, id)
        elif reply == 'Скарга':
            bot.send_message(chat_id, 'Скаргу надіслано')
            sql_user_id=f'SELECT user_id FROM orders WHERE order_id={id}'
            cursor.execute(sql_user_id)
            user_id=cursor.fetchall()[0][0]
            user=int(user_id)
                    sql_volunteer_id=f'SELECT volunteer_id FROM volunteers WHERE tg_volunteer_id={message.chat.id}'
            cursor.execute(sql_volunteer_id)
            volunteer_id=cursor.fetchall()[0][0]
            vol=int(volunteer_id)
            print(f'volunteer_id type: {type(volunteer_id)}')
            print(f'user_id type: {type(user_id)}')
            print(volunteer_id)
            print(user_id)
            process_report_order(id, vol=vol, user=user)
        elif reply == 'В меню':
                                    bot.send_message(chat_id, 'Перехід в меню', reply_markup=types.ReplyKeyboardRemove())
```

```
            process_help(message)
            return
        else:
            raise Exception("oops")
    except Exception as e:
        print(e)
        bot.reply_to(message, 'oooops')


def process_take_order(id: int, volunteer_id: int):
    try:
                sql_take=f"UPDATE orders SET order_status = 'in progress', volunteer_id={volunteer_id} WHERE order_id ={id}"
        cursor.execute(sql_take)
        conn.commit()
    except Exception:
        print("oops")


def process_report_order(id, vol: int, user: int):
    cursor.execute('INSERT INTO reports (volunteer_id, user_id, order_id) VALUES (?, ?, ?)', (vol, user, id))
    cursor.execute(f'UPDATE users SET reports = reports + 1 WHERE id={user}')
    cursor.execute(f'UPDATE orders SET reports = reports + 1 WHERE order_id={id}')
    conn.commit()
```

```
# /my_orders
@bot.message_handler(commands=['my_orders'])
def process_my_orders(message):
    select_orders=f'SELECT orders.order_id, orders.order_desc, orders.order_status, regions.region_description, order_urgency.order_urgency_desc \
   FROM orders JOIN volunteers on orders.volunteer_id = volunteers.volunteer_id JOIN regions on orders.region_id = regions.region_id \
   JOIN order_urgency on orders.order_urgency_id = order_urgency.order_urgency_id \
    WHERE volunteers.tg_volunteer_id = {message.chat.id} and orders.order_status="in progress"'
   cursor.execute(select_orders)
   user_orders=cursor.fetchall()
   if user_orders:
      for order in user_orders:
         bot.send_message(message.chat.id, f'Номер замовлення: {order[0]}\nТекст замовлення: {order[1]}\nСтатус замовлення: {order[2]}\nРегіон замовлення: {order[3]}\nТерміновість: {order[4]}', reply_markup=types.ReplyKeyboardRemove())
   else:
      bot.send_message(message.chat.id, 'У вас немає активних замовлень.', reply_markup=types.ReplyKeyboardRemove())

# /finished_orders
@bot.message_handler(commands=['finished_orders'])
```

```
def process_finished_orders(message):
    select_orders=f'SELECT orders.order_id, orders.order_desc, orders.order_status, regions.region_description, order_urgency.order_urgency_desc \
    FROM orders JOIN volunteers on orders.volunteer_id = volunteers.volunteer_id JOIN regions on orders.region_id = regions.region_id \
    JOIN order_urgency on orders.order_urgency_id = order_urgency.order_urgency_id \
        WHERE volunteers.tg_volunteer_id = {message.chat.id} and orders.order_status="finished"'
    cursor.execute(select_orders)
    user_orders=cursor.fetchall()
    if user_orders:
        for order in user_orders:
            bot.send_message(message.chat.id, f'Номер замовлення: {order[0]}\nТекст замовлення: {order[1]}\nСтатус замовлення: {order[2]}\nРегіон замовлення: {order[3]}\nТерміновість: {order[4]}', reply_markup=types.ReplyKeyboardRemove())
    else:
        bot.send_message(message.chat.id, 'У вас немає завершених замовлень.', reply_markup=types.ReplyKeyboardRemove())


# /close_order
@bot.message_handler(commands=['close_order'])
def close_order(message):
    select_orders=f"SELECT orders.order_id, orders.order_desc, orders.order_status, regions.region_description, order_urgency.order_urgency_desc \
```

```
    FROM orders JOIN volunteers on orders.volunteer_id = volunteers.volunteer_id JOIN regions on orders.region_id = regions.region_id \
    JOIN order_urgency on orders.order_urgency_id = order_urgency.order_urgency_id \
    WHERE volunteers.tg_volunteer_id = {message.chat.id} and orders.order_status='in progress'"
    cursor.execute(select_orders)
    user_orders=cursor.fetchall()
    if not user_orders:
        process_my_orders(message)
        return
    for order in user_orders:
            bot.send_message(message.chat.id, f'Номер замовлення: {order[0]}\nТекст замовлення: {order[1]}\nСтатус замовлення: {order[2]}\nРегіон замовлення: {order[3]}\nТерміновість: {order[4]}')
    msg = bot.reply_to(message, "Введіть номер замовлення, яке хочете завершити.", reply_markup=types.ReplyKeyboardRemove())
    bot.register_next_step_handler(msg, process_close_order_step)


def process_close_order_step(message):
    command = check_command(message)
    if command:
        command(message)
        return
```

```
    select_orders=f'SELECT orders.order_id, orders.order_desc, orders.order_status, regions.region_description, order_urgency.order_urgency_desc \
    FROM orders JOIN volunteers on orders.volunteer_id = volunteers.volunteer_id JOIN regions on orders.region_id = regions.region_id \
    JOIN order_urgency on orders.order_urgency_id = order_urgency.order_urgency_id \
    WHERE volunteers.tg_volunteer_id = {message.chat.id}'
    cursor.execute(select_orders)
    user_orders=cursor.fetchall()
    ids = [i[0] for i in user_orders]
    if message.text.isdigit() and int(message.text) in ids:
        order_delete_dict[message.chat.id] = int(message.text)
        markup = types.ReplyKeyboardMarkup(one_time_keyboard=True)
        markup.add('Так', 'Ні')
        msg = bot.reply_to(message, 'Чи ви підтверджуєте завершення замовлення?', reply_markup=markup)
        bot.register_next_step_handler(msg, process_confirm_close_order_step)
    else:
        msg = bot.reply_to(message, "Введіть правильний номер замовлення.")
        bot.register_next_step_handler(msg, process_close_order_step)


def process_confirm_close_order_step(message):
    command = check_command(message)
    if command:
```

```
        command(message)
        return
    if message.text == 'Так':
        order_id = order_delete_dict[message.chat.id]
         select_user=f'SELECT users.tg_user_id FROM orders JOIN users on orders.user_id = users.id WHERE orders.order_id = {order_id} AND orders.order_status = "in progress"'
        cursor.execute(select_user)
        user=cursor.fetchall()
        print(user)
        if user:
                                select_order=f'SELECT orders.order_id, orders.order_desc, regions.region_description \
            FROM orders JOIN regions on orders.region_id = regions.region_id \
            WHERE orders.order_id = {order_id}'
            cursor.execute(select_order)
            order=cursor.fetchall()
            print(order)
                user_message = f'Волонтер завершив замовлення: \n\nНомер замовлення: {order[0][0]}\nТекст замовлення: {order[0][1]}\nРегіон замовлення: {order[0][2]}'
            user_message_url = urllib.parse.quote(user_message)

requests.get(f'https://api.telegram.org/bot5315016907:AAHzS9bOjvaJ6tGxonTCMH-i_fTaC98sBEc/sendMessage?chat_id={user[0][0]}&text={user_message_url}')
```

```
        delete_order_sql = f'UPDATE orders SET order_status="finished" WHERE order_id = {order_id}'
        cursor.execute(delete_order_sql)
        conn.commit()
        order_delete_dict.pop(message.chat.id)
                                bot.reply_to(message, 'Замовлення завершене.', reply_markup=types.ReplyKeyboardRemove())


# /remove_order
@bot.message_handler(commands=['remove_order'])
def remove_order(message):
        select_orders=f"SELECT orders.order_id, orders.order_desc, orders.order_status, regions.region_description, order_urgency.order_urgency_desc \
    FROM orders JOIN volunteers on orders.volunteer_id = volunteers.volunteer_id JOIN regions on orders.region_id = regions.region_id \
    JOIN order_urgency on orders.order_urgency_id = order_urgency.order_urgency_id \
     WHERE volunteers.tg_volunteer_id = {message.chat.id} and orders.order_status='in progress'"
    cursor.execute(select_orders)
    user_orders=cursor.fetchall()
    if not user_orders:
        process_my_orders(message)
        return
    for order in user_orders:
```

```
            bot.send_message(message.chat.id, f'Номер замовлення: {order[0]}\nТекст замовлення: {order[1]}\nСтатус замовлення: {order[2]}\nРегіон замовлення: {order[3]}\nТерміновість: {order[4]}')

        msg = bot.reply_to(message, "Введіть номер замовлення, від виконання якого хочете відмовитись.", reply_markup=types.ReplyKeyboardRemove())

    bot.register_next_step_handler(msg, process_remove_order_step)


def process_remove_order_step(message):

    command = check_command(message)

    if command:

        command(message)

        return

        select_orders=f'SELECT orders.order_id, orders.order_desc, orders.order_status, regions.region_description, order_urgency.order_urgency_desc \

    FROM orders JOIN volunteers on orders.volunteer_id = volunteers.volunteer_id JOIN regions on orders.region_id = regions.region_id \

    JOIN order_urgency on orders.order_urgency_id = order_urgency.order_urgency_id \

    WHERE volunteers.tg_volunteer_id = {message.chat.id}'

    cursor.execute(select_orders)

    user_orders=cursor.fetchall()

    ids = [i[0] for i in user_orders]

    if message.text.isdigit() and int(message.text) in ids:

        order_delete_dict[message.chat.id] = int(message.text)

        markup = types.ReplyKeyboardMarkup(one_time_keyboard=True)
```

```
        markup.add('Так', 'Ні')
            msg = bot.reply_to(message, 'Чи ви підтвержуєте відміну виконання замовлення?', reply_markup=markup)
        bot.register_next_step_handler(msg, process_confirm_remove_order_step)
    else:
        msg = bot.reply_to(message, "Введіть правильний номер замовлення.")
        bot.register_next_step_handler(msg, process_remove_order_step)


def process_confirm_remove_order_step(message):
    command = check_command(message)
    if command:
        command(message)
        return
    if message.text == 'Так':
        order_id = order_delete_dict[message.chat.id]
         select_user=f'SELECT users.tg_user_id FROM orders JOIN users on orders.user_id = users.id WHERE orders.order_id = {order_id} AND orders.order_status = "in progress"'
        cursor.execute(select_user)
        user=cursor.fetchall()
        print(user)
        if user:
                            select_order=f'SELECT orders.order_id, orders.order_desc, regions.region_description \
```

```
        FROM orders JOIN regions on orders.region_id = regions.region_id \
        WHERE orders.order_id = {order_id}'
        cursor.execute(select_order)
        order=cursor.fetchall()
        print(order)
            user_message = f'Волонтер відмінив виконання замовлення: \n\nНомер замовлення: {order[0][0]}\nТекст замовлення: {order[0][1]}\nРегіон замовлення: {order[0][2]}.\nЙого зможе взяти інший волонтер.'
        user_message_url = urllib.parse.quote(user_message)

requests.get(f'https://api.telegram.org/bot5315016907:AAHzS9bOjvaJ6tGxonTCMH-i_fTaC98sBEc/sendMessage?chat_id={user[0][0]}&text={user_message_url}')
        remove_order_sql = f'UPDATE orders SET order_status="free", volunteer_id = NULL WHERE order_id = {order_id}'
    cursor.execute(remove_order_sql)
    conn.commit()
    order_delete_dict.pop(message.chat.id)
                bot.reply_to(message, 'Виконання замовлення скасоване.', reply_markup=types.ReplyKeyboardRemove())


def check_command(message):
  try:
    command = cmd_list[message.text]
    return command
```

```
    except KeyError:
        return None

#############################################################################
#############################################################################

 #Enable saving next step handlers to file "./.handlers-saves/step.save".
 #Delay=2 means that after any change in next step handlers (e.g. calling register_next_step_handler())
 #saving will hapen after delay 2 seconds.
bot.enable_save_next_step_handlers(delay=2)

cmd_list={'/start': send_welcome, '/help': process_help, '/register': process_register, \
          '/orders': send_order, '/my_orders': process_my_orders, '/finished_orders': process_finished_orders, \
    '/close_order': close_order, '/remove_order': remove_order}
 #Load next_step_handlers from save file (default "./.handlers-saves/step.save")
 #WARNING It will work only if enable_save_next_step_handlers was called!
 #bot.load_next_step_handlers()

if __name__ == '__main__':
    bot.polling(none_stop=True)
```

```
one_time_keyboard=True
```