МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Навчально-науковий інститут бізнесу, економіки та менеджменту
Кафедра економічної кібернетики

«До захисту допущено»

Завідувач кафедри

_____ В. В. Койбічук

«__» _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня ___бакалавра___
(бакалавр / магістр)

зі спеціальності ___051 «Економіка»_____,
(код та назва)

освітньо-професійної програми «Економічна кібернетика та бізнес аналітика»
(освітньо-професійної / освітньо-наукової) (назва програми)

на тему: «Розробка модуля системи автоматизації обробки замовлень клієнтів середнього та малого бізнесу»

Здобувача (ки) групи ЕК-01а Путінцева Анна Віталіївна
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Анна ПУТІНЦЕВА
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник
асистент, PhD, Олександр КУШНЕРЬОВ _____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2024

Міністерство освіти і науки України

Сумський державний університет

Навчально-науковий інститут бізнесу, економіки та менеджменту

Кафедра економічної кібернетики

ЗАТВЕРДЖУЮ
Завідувачка кафедри
доцентка, к.е.н.
_____ В. В. Койбічук
“__” _____ 2024 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА
спеціальність 051 Економіка (Економічна кібернетика та бізнес аналітика)
студентки 4 курсу, групи ЕК-01а

Путінцева Анна Віталіївна

1. Тема роботи «Розробка модуля системи автоматизації обробки замовлень клієнтів середнього та малого бізнесу» затверджена наказом Про затвердження тем і керівників кваліфікаційних робіт наказ №0481-VI від 07.05.2024 року.
2. Термін подання студентом закінченої роботи «1» червня 2024 року.
3. Мета кваліфікаційної роботи: розробка модуля системи автоматизації обробки замовлень клієнтів, який буде простим у використанні, доступним за ціною та відповідати потребам середнього та малого бізнесу.
4. Об’єкт дослідження є процеси прийому замовлень, управління статусом замовлень, інтеграція з існуючими системами, аналітика та звітування.
5. Предмет дослідження є методи автоматизації цих процесів за допомогою розробки Telegram-бота.
6. Кваліфікаційна робота ґрунтується на всебічному підході, який поєднує глибоке вивчення наукових публікацій, аналіз досліджень вітчизняних та зарубіжних авторів з теми автоматизації та обробки замовлень, впровадження систем автоматизації, використання мови Python для проведення досліджень, аналізу даних та моделювання процесів.

7. Орієнтовний план кваліфікаційної роботи, терміни подання розділів керівникові та зміст завдань для виконання поставленої мети

Розділ 1. Огляд існуючих систем управління замовленнями

У розділі 1: 1.1 Аналіз існуючих систем управління замовленнями.

1.2 Аналіз потреб користувачів та бізнесу.

1.3 Технічне завдання до розробки модуля управління онлайн замовленнями .

Розділ 2. Розробка модуля управління онлайн замовленнями

У розділі 2: 2.1Стек технологій які використовувався при розробці модуля.

2.2 Описання структури і функціоналу модуля управління онлайн замовленнями.

2.3 Оцінка очікуваного ефекту від впровадження модуля управлінні онлайн замовленнями.

8. Консультації з роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Кушнерьов О.С., PhD, асистент	01.04.2024	07.05.2024
2	Кушнерьов О.С., PhD, асистент	10.05.2024	27.05.2024

9. Дата видачі завдання: «1» квітня 2024 року

Керівник кваліфікаційної роботи. _____ О.С. Кушнерьов
(підпис) (ініціали, прізвище)

Завдання до виконання одержав. _____ А. В. Путінцева
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

кваліфікаційної роботи на тему
«РОЗРОБКА МОДУЛЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ОБРОБКИ ЗАМОВЛЕНЬ КЛІЄНТІВ СЕРЕДНЬОГО ТА МАЛОГО БІЗНЕСУ »
студентки Путінцева Анна Віталіївна

Актуальність теми, обраної для дослідження. У сучасному динамічному світі середній та малий бізнес стикається з численними викликами, пов'язаними з оптимізацією своїх операцій та підвищенням ефективності. Одним із ключових аспектів, де СМБ може значно покращити свої показники, є автоматизація обробки замовлень клієнтів. Впровадження автоматизації за допомогою месенджера Telegram, може значно покращити рівень обслуговування клієнтів та оптимізувати роботу.

Мета кваліфікаційної роботи полягає в розробці модуля системи автоматизації обробки замовлень клієнтів для середнього та малого бізнесу, створення ефективного та зручного інструменту, який дозволить оптимізувати процеси взаємодії з клієнтами. Цей модуль сприятиме підвищенню продуктивності, зменшенню часу обробки замовлень та підвищенню задоволеності клієнтів. Використання сучасних технологій, таких як чат-боти на платформі Telegram, забезпечить швидкий і зручний доступ клієнтів до послуг бізнесу, що у свою чергу сприятиме збільшенню обсягів продажів і покращенню репутації компанії.

Об'єктом дослідження є процеси прийому замовлень, управління статусом замовлень, інтеграція з існуючими системами, аналітика та звітування за допомогою Telegram.

Предметом дослідження є модуль системи автоматизації обробки замовлень клієнтів, призначений для середнього та малого бізнесу за допомогою розробки Telegram-бота.

Методи дослідження що використовуються для розробки модуля автоматизації обробки замовлень клієнтів, це аналіз літературних джерел та існуючих систем управління замовленнями, аналіз потреб користувачів та

бізнесу, програмна реалізація та тестування бота. Використання Python та фреймворку aiogram для розробки, SQLite для бази даних.

Інформаційна база охоплює вивчення наукової літератури та аналіз поточних методів взаємодії, дослідження можливостей Telegram, а також використання програмних продуктів, таких як Flask, React.js, SQLite, aiogram.

Практичний результат полягає у створенні Telegram-бота, який який дозволяє клієнтам здійснювати замовлення через зручний інтерфейс месенджера. Забезпечено можливість реєстрації користувачів, подання замовлень та відстеження їх статусу через бот.

Теоретичні результати аналіз існуючих систем управління, аналіз потреб користувачів, вивчення можливостей месенджерів, аналіз програмних продуктів, проектування системи автоматизації.

Новизна даної роботи полягає у комплексному підході до автоматизації процесів обробки замовлень у середньому та малому бізнесі за допомогою сучасних технологій та інструментів.

Результати роботи що стосується розробки Telegram-бота для автоматизації обробки замовлень клієнтів СМБ, можуть бути використані для покращення обслуговування клієнтів, скорочення часу на обробку замовлень, скоротити витрати та підвищити задоволеність клієнтів.

Впровадження Telegram-бота, розробленого в рамках кваліфікаційної роботи, може мати значний позитивний вплив на роботу СМБ , що допоможе їм покращити всі аспекти роботи з клієнтами, оптимізувати витрати та стимулювати зростання бізнесу.

Ключові слова: автоматизація, Telegram-бот, управління замовленнями, записи, послуги, чат-бот, Flask, React.js, aiogram, Python.

Зміст кваліфікаційної роботи викладено на 38 сторінках. Список використаних джерел із 35 найменувань, розміщений на 3 сторінках. Робота містить 19 малюнків, 1 формулу, 2 додатки, розміщених на 21 сторінках.

Рік виконання кваліфікаційної роботи – 2024 рік.

Рік захисту роботи – 2024 рік.

ЗМІСТ

ВСТУП

В сучасному світі, де електронна комерція стрімко розвивається, для середнього та малого бізнесу (СМБ) стає все більш важливим мати ефективну систему обробки замовлень клієнтів. Ручне оброблення замовлень може бути трудомістким, схильним до помилок і забирати багато часу, що може призвести до невдоволеності клієнтів та втрати потенційного доходу.

Розробка модуля автоматизації обробки замовлень клієнтів для СМБ може вирішити ці проблеми та допомогти підприємствам покращити свою операційну ефективність, задоволеність клієнтів та загальну рентабельність інвестицій.

На сьогодні грамотна автоматизація бізнес-процесів є ключовим фактором успіху будь-якого підприємства. Новітні інформаційні технології пропонують досконалі методи обробки та аналізу даних, що значно розширюють можливості для ефективного управління.

Сучасний ринок ставить жорсткі вимоги до бізнесу: постійне зростання продуктивності, швидка адаптація до змін, бездоганне обслуговування клієнтів, мінімізація витрат та чітке прогнозування. Для досягнення цих цілей необхідна достовірна інформація для всебічного аналізу: стан виробництва, товарні запаси, відносини з постачальниками та філіями, управління персоналом, документообіг.

Саме тут на допомогу приходять інформаційні системи. Вони автоматизують рутинні задачі, економлять час та ресурси, надають доступ до актуальних даних для прийняття обґрунтованих рішень. Автоматизація стає неминучим кроком у розвитку будь-якого бізнесу, адже шляхи та методи можуть відрізнятися, але мета одна - оптимізація роботи та підвищення конкурентоспроможності.

Метою дослідження роботи є розробка модуля автоматизації обробки замовлень клієнтів для СМБ, який буде простим у використанні, ефективним та доступним, а саме розробка Telegram-боту.

Для реалізації поставленої мети важливо виконати наступні завдання:

– провести аналіз існуючих систем управління замовленнями та визначити їхні сильні та слабкі сторони;

– проаналізувати потреби користувачів та бізнесу, щоб визначити вимоги до модуля автоматизації обробки замовлень;

– розробити технічне завдання для створення Telegram-бота, включаючи вибір технологічного стека, функціональні можливості та архітектуру.

– спроектувати та реалізувати Telegram-бот для автоматизації обробки замовлень.

– оцінити ефективність модуля та його вплив на операційну діяльність СМБ.

Об'єкт дослідження є система автоматизованої обробки замовлень клієнтів для СМБ. Ця система може включати різні компоненти, такі як програмне забезпечення, апаратне забезпечення та людські ресурси.

Предмет дослідження є розробка модуля автоматизації обробки замовлень клієнтів для СМБ. Цей модуль є одним із компонентів системи автоматизованої обробки замовлень клієнтів і буде досліджуватися більш детально.

Результати роботи є впровадження Telegram-бота, розробленого в рамках цієї роботи, може стати вигідною інвестицією для СМБ, що допоможе їм покращити всі аспекти роботи з клієнтами, оптимізувати витрати та стимулювати зростання бізнесу.

Розробка Telegram-бота для автоматизації обробки замовлень клієнтів СМБ ґрунтувалась на наступному наборі програмних продуктів: AdminPane – це потужна система управління, розроблена для спрощення та оптимізації процесів розробки та адміністрування мобільних додатків. React – JavaScript-бібліотека для створення динамічних інтерфейсів користувача. React використовується для побудови зручного та інтуїтивно зрозумілого інтерфейсу AdminPane, що робить роботу з системою простою та приємною. FullCalendar – потужний JavaScript-компонент календаря, який

використовується для візуалізації, пов'язаних з розробкою мобільного додатку. SweetAlert2 – JavaScript-бібліотека для створення інформативних спливаючих вікон, які використовуються в AdminPane Axios – JavaScript-бібліотека для HTTP-запитів, яка використовується для зв'язку AdminPane з бекенд-сервером. Python-dotenv – бібліотека Python, яка використовується для безпечного завантаження змінних середовища з файлу .env. Aiogram – потужна Python-бібліотека для розробки ботів Telegram. Flask – легкий і гнучкий веб-фреймворк Python, який використовується для створення API AdminPane. Flask забезпечує простий та інтуїтивно зрозумілий інтерфейс для розробки API, що робить його ідеальним вибором для AdminPane.

1 ОГЛЯД ТЕОРЕТИЧНИХ ОСНОВ РОЗРОБКИ МОДУЛЯ УПРАВЛІННЯ ОНЛАЙН ЗАМОВЛЕННЯМИ

1.1 Аналіз існуючих систем управління замовленнями

Сучасний ринок пропонує широкий спектр систем управління замовленнями (СУЗ), що відрізняються функціональністю, ціною, складністю впровадження та масштабованістю.

За типом розміщення СУЗ можна поділити на:

– Локальні: встановлюються на сервері замовника та потребують власного IT-персоналу для адміністрування.

– Хмарні: доступні через інтернет, не потребують локальної інфраструктури та адміністрування.

За функціональністю СУЗ можна класифікувати на:

- WMS (Warehouse Management System): системи управління складом, що фокусуються на управлінні складськими запасами, прийомом, розміщенням, відбором та відвантаженням товарів.
- OMS (Order Management System): системи управління замовленнями, що обробляють замовлення від клієнтів, відстежують їх статус, інтегруються з платіжними системами та службами доставки.
- CRM (Customer Relationship Management): системи управління взаємовідносинами з клієнтами, що включають функціонал OMS, а також інструменти для маркетингу, продажів та обслуговування клієнтів.

При виборі СУЗ важливо врахувати такі фактори:

- Масштабність: чи може система рости разом з вашим бізнесом?
- Функціональність: чи відповідає система вашим потребам?
- Інтеграція: чи може система інтегруватися з вашими існуючими системами?
- Вартість: яка ціна ліцензії та впровадження?

- Простота використання: чи легко навчити персонал користуватися системою?

Ринок пропонує широкий спектр систем управління замовленнями (СУЗ), кожна з яких має свої особливості та переваги. Попри певні відмінності, більшість СУЗ об'єднує схожий інтерфейс та базовий набір функцій, адже їх головне завдання - ефективне управління даними.

В рамках цього аналізу ми зосередимося на зручності використання та привабливості інтерфейсу СУЗ, адже саме ці аспекти суттєво впливають на загальний досвід користувачів.

Перша система – це « SAP Business One» (рис. 1.1) [27].

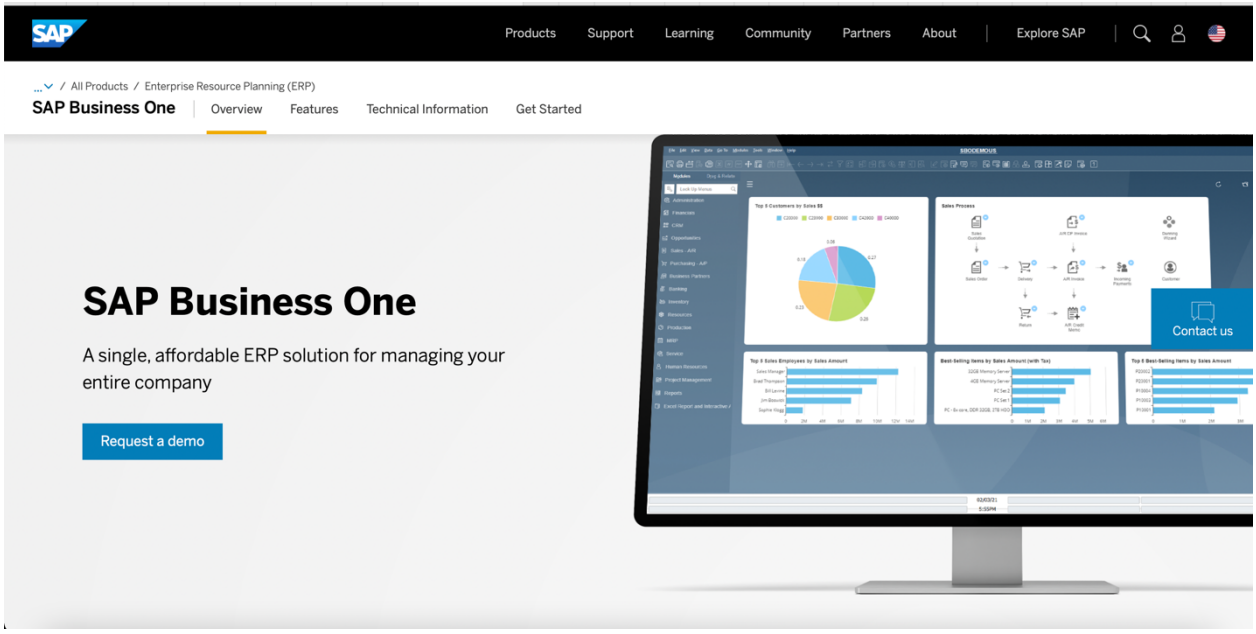

Рисунок 1.1 – Головна сторінка « SAP Business One »

Ця популярна система пропонує широкий спектр функцій, включаючи управління запасами – моніторинг рівня запасів, автоматичне створення замовлень на поповнення, оптимізація складських запасів, управління ланцюжком поставок – відстеження замовлень постачальників, моніторинг рівня обслуговування, оптимізація маршрутів доставки, CRM – управління контактами, відстеження угод, управління продажами та фінансову звітність – створення звітів про прибутки та збитки, балансовий звіт, звіт про рух готівки.

Вона підходить для малого та середнього бізнесу. Але є недоліки використання цієї системи, вона може бути дорогою для малого бізнесу та потребує деяких технічних знань для налаштування та використання.

Друга система – «Sales Creatio». Цей продукт пропонує комплексну автоматизацію та прискорення всього циклу продажів, починаючи від першого контакту з потенційним клієнтом (лідом) і до повторних замовлень. Окрім цього, система пропонує конфігурації для управління маркетингом, сервісним обслуговуванням та іншими бізнес-процесами.рис. (1.2) [8].

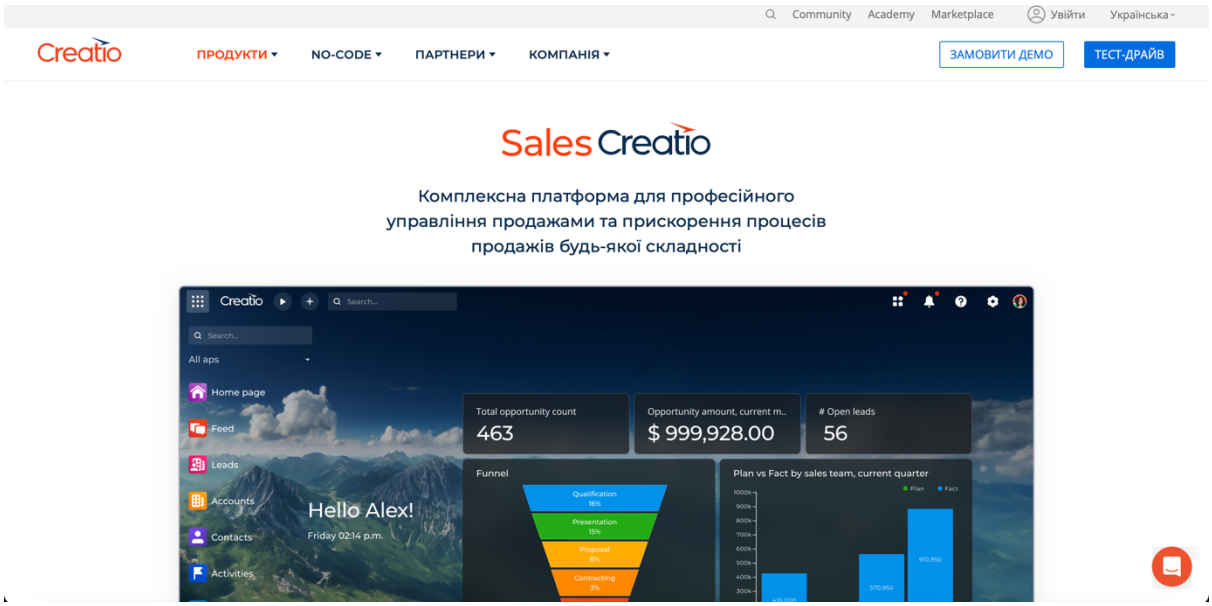

Рисунок 1.2 – Головна сторінка «Sales Creatio»

Marketplace пропонує широкий спектр готових доповнень, галузевих рішень, коннекторів та шаблонів, що дозволяють розширити можливості платформи та автоматизувати різні бізнес-задачі.

Однією з ключових переваг цієї системи є можливість створити демоверсію, навіть незважаючи на те, що вона є платною. Створення демоверсії є простим та швидким процесом. Вам просто потрібно зареєструватися на веб-сайті та вказати свою контактну інформацію. Після

реєстрації ви отримаєте доступ до демоверсії, яка буде доступна протягом певного періоду часу. (рис. 1.3) [8].

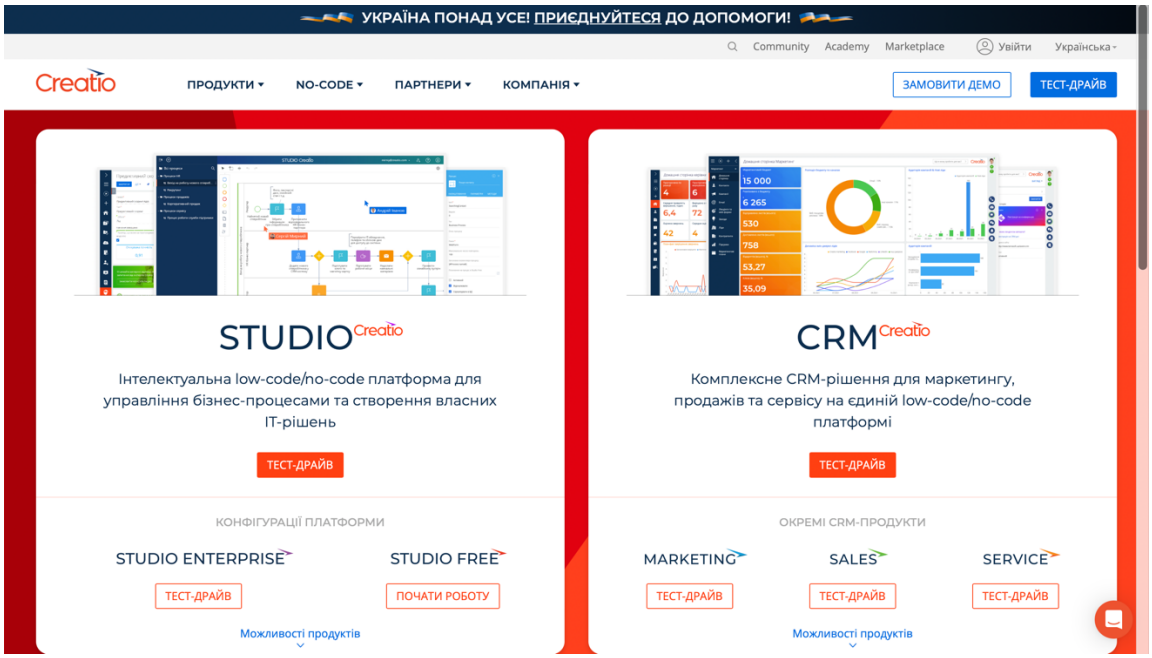

Рисунок 1.3 – Реєстрація демо-версії «Sales Creatio»

Окрім можливості протестувати систему перед покупкою, демоверсія також наповнена демо-даними, що дозволяє вам краще уявити її роботу. Функції, які доступні в демоверсії: управління замовленнями та рахунками, управління продажам, управління продуктами, управління документами, розклад та комунікації, планування та аналіз продажів, інструменти налаштування [8].

Третя система – «Oracle Order Management». Ця система пропонує широкий спектр функцій для управління замовленнями, включаючи автоматизовану обробку, відстеження поставок та управління інвентарем. Oracle Order Management може інтегруватися з іншими системами Oracle, такими як управління складом та фінанси. Вартість використання Oracle Order Management може бути високою, особливо для малих підприємств. Проте Oracle також пропонує хмарні рішення, які можуть бути доступні для менших бізнесів. Oracle Order Management може бути масштабованим для великих підприємств з високими обсягами замовлень. (рис. 1.4) [22].

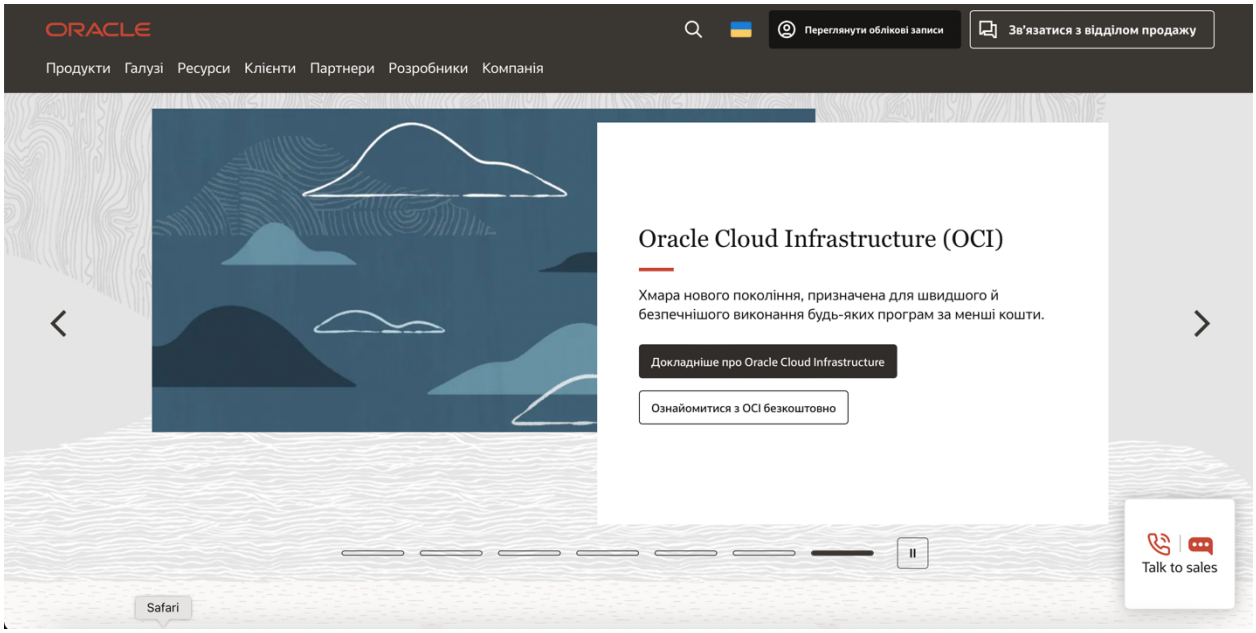

Рисунок 1.4 – Головна сторінка «Oracle Order Management»

Четверта система – «Magento». Це потужна платформа електронної комерції, яка має вбудовану систему управління замовленнями. Вона дозволяє ефективно обробляти замовлення, відстежувати їх статуси, керувати запасами та взаємодіяти з клієнтами. Magento має можливості для інтеграції з різними сторонніми системами, такими як системи CRM, платіжні шлюзи та системи управління запасами. Вартість Magento може бути більш доступною порівняно з деякими іншими системами управління замовленнями, особливо для малих та середніх підприємств. Проте існують витрати на розробку, налаштування та підтримку. Magento може бути використаний як для невеликих інтернет-магазинів, так і для великих електронних торгових платформ (рис. 1.5) [9; 29].

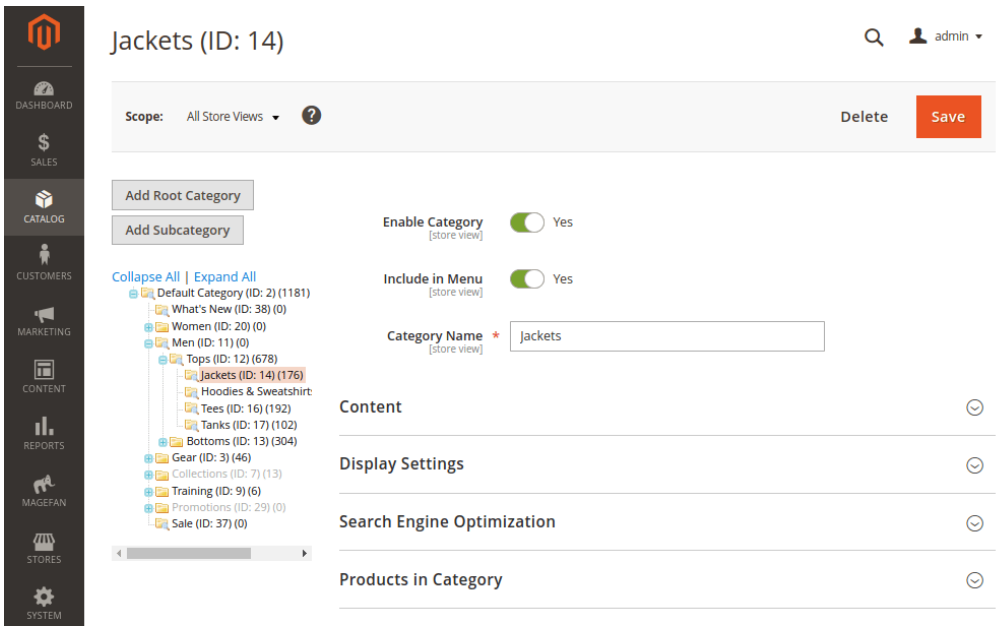

Рисунок 1.5 – Інтерфейс «Magento»

П’ята система – «Shopify». Це популярна платформа електронної комерції, яка має у власній системі ефективне управління замовленнями. Вона пропонує інтуїтивний і простий у використанні інтерфейс для обробки замовлень, відстеження статусів і спілкування з клієнтами. Shopify також має можливості для інтеграції з різними додатками та сторонніми системами, які дозволяють розширити функціональність вашого магазину. Shopify може бути доступним для різних бюджетів, залежно від обраного плану. Вартість включає плату за користування платформою, а також витрати на додаткові додатки та розширення. Shopify підходить для малих, середніх і великих інтернет-магазинів і може бути легко масштабованим з ростом вашого бізнесу (рис. 1.6) [11; 28].

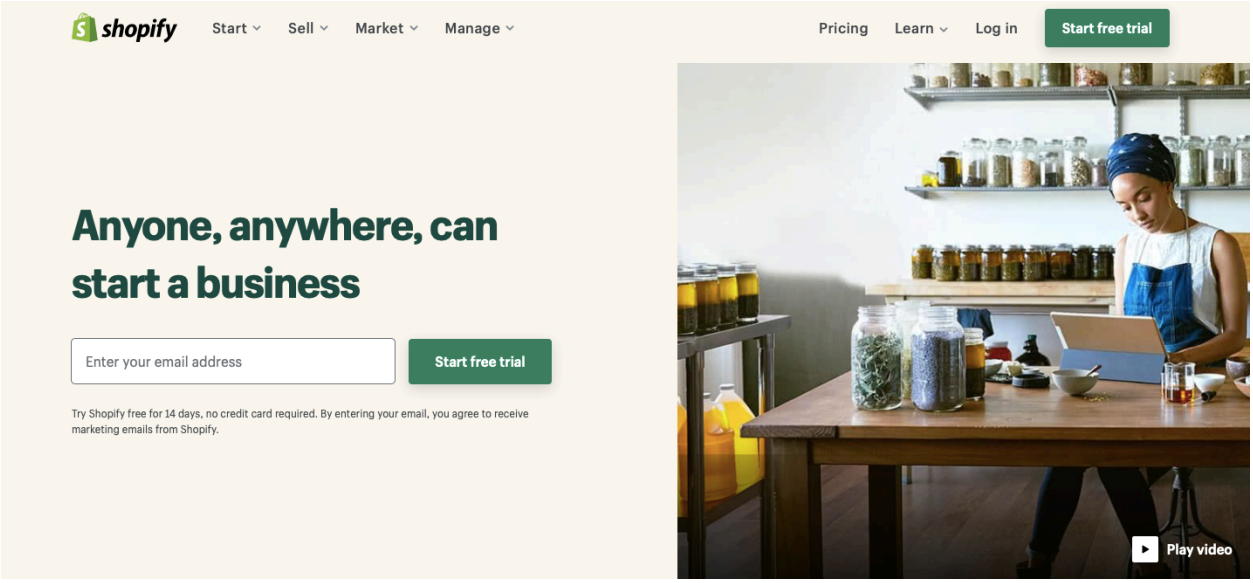

Рисунок 1.6 – Головна сторінка « Shopify »

Кожна з цих систем має свої особливості, і правильний вибір буде залежати від конкретних потреб вашого бізнесу, бюджету та стратегії розвитку.

1.2 Аналіз потреб користувачів та бізнесу

Аналіз потреб користувачів та бізнесу є ключовим етапом у визначенні вимог до системи управління замовленнями. Ось деякі аспекти, які можна врахувати під час цього аналізу:

1. Потреби користувачів:

– Зручність інтерфейсу: Користувачі можуть бажати простого та інтуїтивно зрозумілого інтерфейсу для розміщення та відстеження замовлень.

– Мобільність: У світі, де мобільність важливіша, користувачі можуть очікувати можливості робити замовлення з мобільних пристроїв.

– Забезпечення конфіденційності: Забезпечення безпеки та конфіденційності особистої інформації клієнтів є надзвичайно важливим аспектом для багатьох користувачів. Особливо це стосується платіжних даних.

– Підтримка клієнтів: Користувачі можуть бажати доступу до служби підтримки, яка допоможе їм у разі виникнення питань або проблем[35].

2. Потреби бізнесу:

– Автоматизація процесів: Бізнес може шукати систему, яка допоможе автоматизувати процеси прийому та обробки замовлень, щоб зменшити людський фактор та підвищити ефективність.

– Управління запасами: Важливим аспектом для бізнесу є можливість ефективного керування інвентарем, щоб уникнути проблем з недостатньою або надмірною кількістю товарів.

– Аналітика та звітність: Бізнес може бажати мати доступ до аналітичної інформації про замовлення, прибутковість, поведінку клієнтів тощо, щоб приймати обґрунтовані рішення.

– Масштабованість: Система повинна бути готовою масштабуватися разом з ростом бізнесу та збільшенням обсягу замовлень.

Аналіз цих потреб допоможе визначити функціональні вимоги до системи управління замовленнями, які відповідають потребам користувачів та бізнесу. Також це допоможе забезпечити успішну реалізацію та прийняття системи всіма зацікавленими сторонами [20].

1.3 Технічне завдання до розробки модуля управління онлайн замовленнями

Для системи управління онлайн замовленнями, потрібна зручна система, завдяки якій можна резервувати дати на певні послуги, редагувати підтверджувати записи, вести клієнтську базу та інше. В зручності можна створити мобільний додаток, але є деталі незручності, наприклад проблеми з установкою мобільного додатку або з реєстрацією. А з іншого боку, ми можемо використати телеграм(чат-бот), який зручний, більшість користувачів користуються цим месенджером, і це допоможе нам розробці нашої системи[2; 12; 19].

Метою розробки модуля є автоматизація процесу прийому замовлень на послуги та запису клієнтів, а також покращення обслуговування клієнтів за допомогою чат-бота, що призведе до:

- Підвищення ефективності: скорочення часу обробки замовлень, зменшення рутинних завдань,
- Покращення обслуговування клієнтів: швидкий та зручний запис на послуги, цілодобова підтримка,
- Збільшення продажів: стимулювання повторних записів, персоналізовані рекомендації.

Опис функціональності:

1. Реєстрація клієнтів через чат-бот:

– Клієнти можуть реєструватися за допомогою чат-бота, надавши необхідні особисті дані, такі як ім'я, прізвище, номер телефону тощо.

2. Запис на послуги через чат-бот:

– Клієнти можуть записуватися на послуги за допомогою чат-бота, вибираючи дату та час з допомогою календаря, який надається.

3. Підтвердження та скасування запису:

– Клієнти можуть отримувати підтвердження про успішний запис або скасування запису за допомогою повідомлень від чат-бота.

4. Нагадування про запис:

– Система повинна автоматично нагадувати клієнтам про найближчі записи за певний період до них.

Технічні вимоги:

1. Frontend (Admin Panel):

Розробка адміністративної панелі:

– Розробка веб-інтерфейсу за допомогою бібліотеки React.js, яка надає можливість адміністратору керувати записами та клієнтами.

– Створення різних компонентів, таких як таблиці для відображення списку записів та клієнтів, форми для додавання нових записів або редагування інформації про клієнтів.

– Використання бібліотеки FullCalendar для відображення календаря з записаними подіями, такими як дати та часи записів клієнтів.

Відображення сповіщень:

– Використання бібліотеки SweetAlert2 для відображення сповіщень про результати виконаних дій, таких як успішне додавання нового запису або підтвердження видалення запису.

2. Backend (API):

Розробка API:

– Розробка RESTful API за допомогою фреймворку Flask у Python для обробки запитів від чат-бота та адміністративної панелі.

– Визначення різних маршрутів API для обробки запитів, таких як створення нових записів, отримання списку існуючих записів та клієнтів, оновлення даних профілю клієнта тощо.

Управління базою даних:

– Використання бази даних для збереження інформації про замовлення та клієнтів. Можливо, використання ORM (Object-Relational Mapping) для спрощення взаємодії з базою даних.

3. Чат-бот:

Реалізація бота:

– Використання бібліотеки aiogram для розробки чат-бота в месенджері, такому як Telegram або інший.

– Надання можливості клієнтам реєструватися та записуватися на послуги через взаємодію з ботом.

Взаємодія з бекендом:

– Реалізація зв'язку між чат-ботом та бекендом через API для обміну даними про клієнтів та їх записами.

4. Безпека та інші вимоги:

Захист даних:

– Забезпечення безпеки особистих даних клієнтів, використовуючи шифрування та інші методи захисту даних.

– Використання HTTPS для забезпечення безпеки під час передачі даних між клієнтом та сервером.

Управління конфігурацією:

– Використання бібліотеки python-dotenv для керування конфігурацією середовища у Python.

2 РОЗРОБКА МОДУЛЯ УПРАВЛІННЯ ОНЛАЙН ЗАМОВЛЕННЯМИ

2.1 Стек технологій які використовувався при розробці модуля

У сучасному світі компаніям доводиться постійно пристосовуватися до змін у технологіях, щоб залишатися конкурентоспроможними. Особливо важливо це для малого та середнього бізнесу, де використання новітніх інструментів може забезпечити перевагу на ринку.

Розробка Telegram-бота для автоматизації обробки замовлень є одним із способів використання сучасних технологій для поліпшення бізнес-процесів. Цей бот спрощує процес запису для клієнтів і дозволяє менеджеру ефективно взаємодіяти з клієнтами та швидко реагувати на їхні потреби.

Telegram-бот був розроблений в Python. Вибір Python для розробки Telegram-бота ґрунтується на низці вагомих аргументів:

1. Універсальність: Python – це мова програмування загального призначення, що робить її придатною для широкого кола завдань, зокрема й розробки Telegram-ботів. Її універсальність дає змогу використовувати Python у різних проектах, не обмежуючись лише сферою ботів.

2. Простота та лаконічність: Python вирізняється чітким та читабельним кодом, який легко вивчати та використовувати навіть початківцям. Ця властивість робить розробку Telegram-бота на Python доступною для ширшого кола людей, не вимагаючи поглиблених знань у програмуванні.

3. Продуктивність: Python дозволяє швидко писати код, економлячи час та зусилля розробників. Завдяки лаконічному синтаксису та чіткій структурі коду, Python дає змогу створювати Telegram-боти швидше, ніж при використанні деяких інших мов програмування.

4. Кросплатформність: Програми Python без проблем запускаються на різних операційних системах, таких як Windows, macOS та Linux. Це робить Telegram-ботів, розроблених на Python, доступними для користувачів різних платформ, не потребуючи додаткових зусиль з адаптації.

5. Багата бібліотека: Python постачається з великою стандартною бібліотекою, яка містить багато корисних інструментів для веб-розробки, роботи з даними, обробки текстів та багато іншого. Цей широкий набір інструментів полегшує розробку Telegram-бота, надаючи готові рішення для типових завдань.

6. Масштабованість: Python підходить як для створення невеликих скриптів, так і для розробки складних систем. Це робить його гнучким вибором для проектів будь-якого розміру. Telegram-боти, розроблені на Python, можуть легко розширюватися та доповнюватися новими функціями в міру зростання потреб.

Зважаючи на перелічені вище переваги, Python стає чудовим вибором мови програмування для розробки Telegram-бота. Його універсальність, простота, продуктивність, кросплатформність, багата бібліотека та масштабованість роблять його зручним інструментом для створення ефективних та надійних Telegram-ботів [4;23].

Python – це інтерпретована мова програмування, що означає, що код виконується по рядку. Код виконується по рядку, а не компілюється в машинний код. Це робить його повільнішим за компільовані мови, але дає ряд переваг. При виявленні помилки, виконання зупиняється, і повідомляється про її місцезнаходження. Це спрощує налагодження, адже ви можете зосередитися на одній помилці за раз.Загалом, інтерпретована природа Python робить його зручним для швидкого розробки та налагодження програм, хоча й трохи менш продуктивним, ніж компільовані мови.

Python має ліцензію з відкритим кодом, схвалену OSI, що робить його: безкоштовним, відкритим та гнучким [10].

Python особливо цінна тим, що крім великої стандартної бібліотеки надає величезний набір додаткових модулів, розроблених спеціально для аналітичних цілей. Найвідоміші бібліотеки Python для аналізу даних — це pandas і NumPy . Ці інструменти дозволяють робити з вашими даними майже

все, наприклад, очищати і аналізувати їх, вивчати статистику або візуалізувати приховані тенденції у ваших даних.

Python набув широкої популярності в останні роки завдяки своїй універсальності та простоті використання. Його застосовують у різних галузях, від машинного навчання до веб-розробки та тестування програмного забезпечення. Ця мова доступна як для досвідчених розробників, так і для фахівців без попереднього досвіду програмування. Її простий синтаксис та чітка структура роблять Python легким для вивчення та використання [5].

Для розробки Telegram-бота було обрано середовище Node.js від JavaScript. Node.js – це середовище виконання JavaScript з відкритим кодом, що використовує рушій V8 від Google. Це робить Node.js:

- Швидким: V8 оптимізований для JavaScript, що робить Node.js одним з найшвидших середовищ виконання.
- Ефективним: Node.js використовує асинхронну модель вводу/виводу, що дозволяє йому обробляти багато запитів одночасно.
- Гнучким: Node.js можна використовувати для створення веб-серверів, фреймворків, інструментів для резервного копіювання та багато іншого.

Node.js також має пакетний менеджер npm, який полегшує встановлення та керування залежностями для ваших проектів [26].

Node.js ґрунтується на асинхронній моделі програмування, що робить його ідеальним для:

- Мережевих серверів: Node.js може обробляти багато одночасних запитів без блокування процесу, що робить його чудовим вибором для розробки мережевих серверів.
- Масштабованих застосунків: Асинхронна природа Node.js дозволяє йому ефективно масштабуватися на велику кількість користувачів та запитів.

- Ресурсомістких програм: Node.js економить ресурси, оскільки не витрачає час на очікування завершення операцій вводу/виводу [26].

Розробники можуть використовувати JavaScript як для фронт-енду (інтерактивних частин веб-сайту, які бачить користувач), так і для бек-енду (серверної частини, яка обробляє дані та логіку). Це спрощує розробку та підтримку коду, адже не потрібно перемикатися між різними мовами [21].

Для створення Telegram-боту використовувалась бібліотека Aiogram. Це популярна бібліотека Python, спеціально розроблена для створення Telegram-ботів. Вона використовує фреймворк asyncio для створення асинхронних ботів, що дозволяє їм ефективно обробляти декілька запитів одночасно.

Aiogram використовує asyncio, що дозволяє вашому боту обробляти одночасно численні взаємодії з користувачами. Це критично важливо для ботів, які очікують високого трафіку, адже це гарантує швидкість реагування та плавний досвід користувача.

Aiogram надає чітке та лаконічне API для взаємодії з Telegram Bot API. Це робить розробку швидшою та простішою у підтримці, особливо порівняно з написанням необроблених викликів API [13].

Aiogram пропонує широкий спектр функцій, які охоплюють всі ключові аспекти розробки ботів. До них належать:

– Взаємодія з користувачем: обробка повідомлень, зворотних дзвінків, вбудованих запитів та опитувань.

– Управління даними: зберігання та отримання даних за допомогою різних методів, таких як зберігання в пам'яті або бази даних.

– ПЗП: реалізація власної логіки для перехоплення та зміни поведінки бота на різних етапах.

– Диспетчер: маршрутизація вхідних оновлень до відповідних обробників для ефективної обробки.

– Робота з файлами: завантаження та завантаження файлів для різних функціональних можливостей бота.

– Створення клавіатур: створення власних клавіатур з кнопками для взаємодії з користувачем.

Aiogram без проблем інтегрується з іншими бібліотеками Python, що дозволяє розширювати можливості вашого бота. Наприклад, ви можете інтегрувати бібліотеки баз даних для постійного зберігання даних або веб-фреймворки для створення веб-інтерфейсів для адміністрування бота [13].

Загалом, Aiogram – це потужна та зручна бібліотека, яка спрощує розробку Telegram-ботів. Її асинхронна природа, чистий синтаксис та багатий функціонал роблять її чудовим вибором для створення гнучких та багатофункціональних ботів.

Для створення інтерфейсу адміністратора, який буде використовуватися для керування записами, застосовано React JS. React – це популярна JavaScript-бібліотека, призначена для розробки динамічних та інтерактивних веб-інтерфейсів. Її переваги:

- Відкритий код: React є безкоштовним та загальнодоступним для використання.
- Простота у вивченні: React має простий синтаксис та чітку структуру, що робить його легким для вивчення.
- Ефективність: React використовує віртуальний DOM, який робить його надзвичайно ефективним при рендерингу інтерфейсів.
- Масштабованість: React можна використовувати для створення складних та масштабованих веб-додатків.
- Компонентний підхід: React використовує компоненти, які роблять код модульним та повторно використовуваним.

Завдяки своїм перевагам React став чудовим вибором для розробки інтерфейсів користувача веб-додатків.

У контексті панелі адміністрування React буде використовуватися для:

- Створення динамічних форм для додавання та редагування записів.
- Відображення таблиць з даними записів.

- Забезпечення інтерактивності для керування записами.

В цілому React JS дозволить створити зручний та ефективний інтерфейс адміністрування для керування записами [24; 25].

React JS чудово підходить для розробки проектів різної складності завдяки:

- Модульності: Компонентний підхід React робить код модульним та організованим, що полегшує його підтримку та розширення.
- Гнучкості: React можна інтегрувати з іншими бібліотеками та фреймворками, що дає розробникам більше можливостей.
- SEO-оптимізації: React підтримує серверний рендеринг, який покращує швидкість завантаження сторінок та оптимізує SEO [24].

Для того щоб відображались записи у форматі календаря використовувався FullCalendar.io. Адміністратори можуть переглядати майбутні записи, фільтрувати за датою/користувачем і, можливо, керувати записами безпосередньо в календарі [17].

Для відображення повідомлень використовувалась SweetAlert2. Це JavaScript–бібліотека для створення модальных вікон з різними функціональними можливостями, такими як [31]:

- Інформаційні повідомлення: Ви можете використовувати SweetAlert2, щоб відображати прості повідомлення з текстом та іконкою.
- Підтвердження: Ви можете створювати модальні вікна з кнопками "Підтвердити" та "Скасувати", щоб отримати згоду користувача на певну дію.
- Введення даних: SweetAlert2 може використовуватися для створення модальных вікон з полями вводу, де користувачі можуть вводити текст або вибирати варіанти з списку.
- Завантаження: Ця бібліотека може візуалізувати процес завантаження з анімацією та індикатором прогресу.

- Налаштування: SweetAlert2 можна налаштувати за допомогою різних параметрів, таких як дизайн, розмір, анімація, звукові ефекти та поведінка кнопок.

Для розробки бек-енду була використана технологія Flask. Ця Python фреймворк веб-додатків буде забезпечувати роботу бек-енд API, який обробляє керування даними та запити користувачів. API [33]:

– Взаємодіятиме з базою даних: Вам знадобиться база даних для зберігання інформації про записи, даних користувачів та, можливо, інших відповідних даних. Flask може інтегруватися з бібліотеками, такими як SQL, для ефективної взаємодії з базою даних.

– Обробка запитів з фронт-енду: API отримуватиме запити від React-фронту через Axios. Ці запити можуть включати отримання даних про записи за певний діапазон дат, оновлення деталей запису, керування обліковими записами користувачів або, можливо, інші функціональні можливості залежно від потреб вашої системи.

– Відповідати на запити: На основі отриманих запитів API буде запитувати базу даних, виконувати необхідні дії (наприклад, оновлення запису) та надсилати відповіді назад на фронт-енд. Ці відповіді можуть бути даними про записи, підтверджуючими повідомленнями або повідомленнями про помилки.

Технологія Axios була використана для зв’язку між між фронт-ендом і бек-ендом. Ця JavaScript-бібліотека буде обробляти зв'язок між React-фронтом та Flask-бек-ендом API. Axios дозволяє фронт-енду робити HTTP-запити до API, наприклад, отримувати дані про записи, оновлювати записи або керувати інформацією про користувачів. API відповідатиме на запит даними або підтвердженням успішних дій [18].

Важливим етапом у створенні бота є розробка структури бази даних. Саме вона буде зберігати та організовувати всю інформацію, необхідну для його роботи.

Для зберігання даних Telegram-бота буде використовуватися SQLite. Ця база даних легко інтегрується з Python, що робить її зручним вибором для цього проекту. SQLite є легкою, вбудованою реляційною системою керування базами даних (СУБД) з відкритим кодом. Вона відрізняється від більшості СУБД тим, що не використовує окремий серверний процес. Натомість, SQLite реалізовано як бібліотеку, яку можна інтегрувати у вашу програму. Це робить її ідеальним рішенням для вбудованих систем, мобільних додатків та інших програм, де потрібна легка та портативна база даних. SQLite має дуже малий розмір файлу (менше 500 КБ), що робить її ідеальною для вбудованих систем та пристроїв з обмеженими ресурсами. SQLite не потребує окремого серверного процесу. Замість цього, вона інтегрується безпосередньо у вашу програму. SQLite має репутацію дуже надійної та стабільної СУБД. Вона використовується у багатьох критично важливих програмах. SQLite підтримує більшість основних операторів та функцій стандарту SQL, що робить її легкою для використання розробниками, які вже знайомі з SQL. SQLite є безкоштовною для використання та розповсюдження під ліцензією громадського надбання. SQLite часто використовується в пристроях з обмеженими ресурсами, таких як маршрутизатори, смартфони та інші електронні пристрої. Багато мобільних додатків використовують SQLite для зберігання локальних даних, таких як контакти, налаштування користувачів та історію. SQLite можна використовувати в настільних програмах для зберігання локальних даних, таких як бази даних клієнтів або кешування даних. Хоча SQLite не призначена для основних веб-додатків, її можна використовувати для зберігання локальних даних у веб-браузері за допомогою веб- сховища (Web Storage). В цілому, SQLite є потужною та універсальною СУБД, яка ідеально підходить для багатьох програм [32].

2.2 Описання структури і функціоналу модуля управління онлайн замовленнями

Створюючи Telegram-бота, на першому етапі розробили базу даних під назвою «database.db». Вона містить чотири таблиці (рис. 2.1) [15].

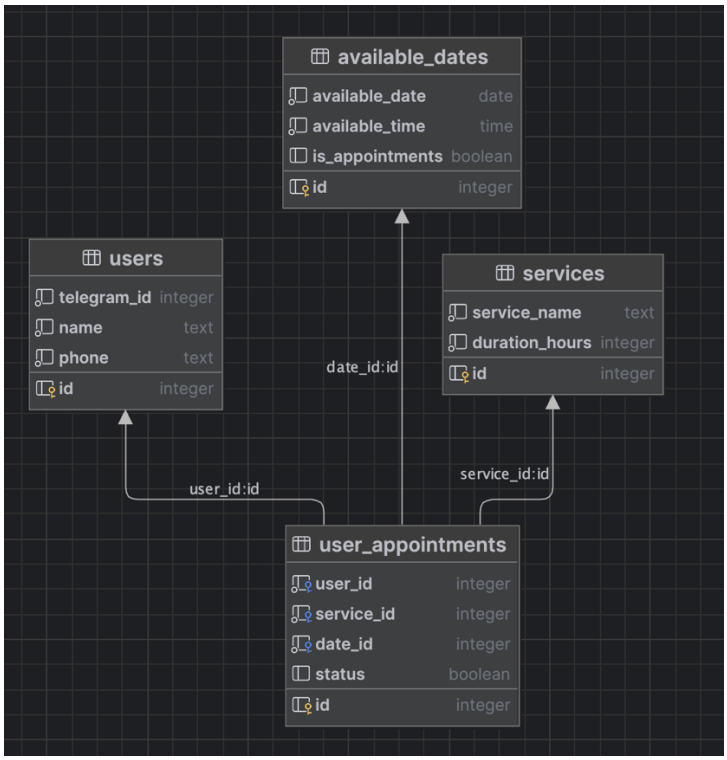

Рисунок 2.1 – Структура бази даних

Цей код SQL створює чотири таблиці для бази даних Telegram-бота:

1. Services: Ця таблиця зберігає інформацію про послуги, що пропонуються ботом, включаючи їх назву та тривалість у годинах.

2. Available_dates: Ця таблиця зберігає інформацію про вільні дати та години для запису на послуги. Кожен запис містить дату, час та прапор, який вказує, чи є цей час вільним.

3. Users: Ця таблиця зберігає інформацію про користувачів бота, включаючи їх Telegram ID, ім'я та номер телефону.

4. User_appointments: Ця таблиця зберігає інформацію про записи користувачів на послуги. Кожен запис містить ID користувача, ID послуги, ID дати та часу, а також статус запису.

Всі таблиці мають первинний ключ (id), який використовується для їх однозначної ідентифікації. Таблиці (user_appointments) містить зовнішні ключі, які пов'язують її з таблицями (users), (services та available_dates). Це забезпечує зв'язок між записами та відповідними користувачами, послугами та вільними часами.

Поле (is_appointments) в таблиці (available_dates) використовується для позначення того, чи є час вільним для запису. Поля (telegram_id та name) додані до таблиці (users) для зберігання додаткової інформації про користувачів. У таблиці (available_dates) додано унікальний ключ для поєднання (available_date та available_time). Це гарантує, що в один і той же час не може бути записано більше одного запису.

Ця структура бази даних дозволяє Telegram-боту ефективно зберігати та керувати інформацією про послуги, користувачів, записи та доступність.

Для створення Telegram-боту використовувався BotFather. Це спеціальний бот у Telegram, який допомагає користувачам створювати власних ботів. Він спрощує процес розробки, надаючи інструменти для налаштування імені, команд та інших параметрів бота.

Ось основні моменти про BotFather:

– Створення бота: BotFather робить створення Telegram-бота доступним для будь-кого, навіть без досвіду програмування.

– Налаштування: BotFather дозволяє присвоїти боту ім'я, опис та список команд, які він буде розуміти та виконувати.

– Простота використання: Взаємодія з BotFather відбувається за допомогою простих команд, що робить процес створення бота інтуїтивно зрозумілим.

– Потужність: BotFather – це лише перший крок. Після його використання ви можете додати боту складні функції за допомогою Python та API Telegram.

Для створення боту потрібно ввести команду «/newbot» до BotFather, він попросить вас придумати ім'я для вашого бота, потім надасть токен бота, який буде потрібен для доступу до бота з кодом (рис. 2.2)

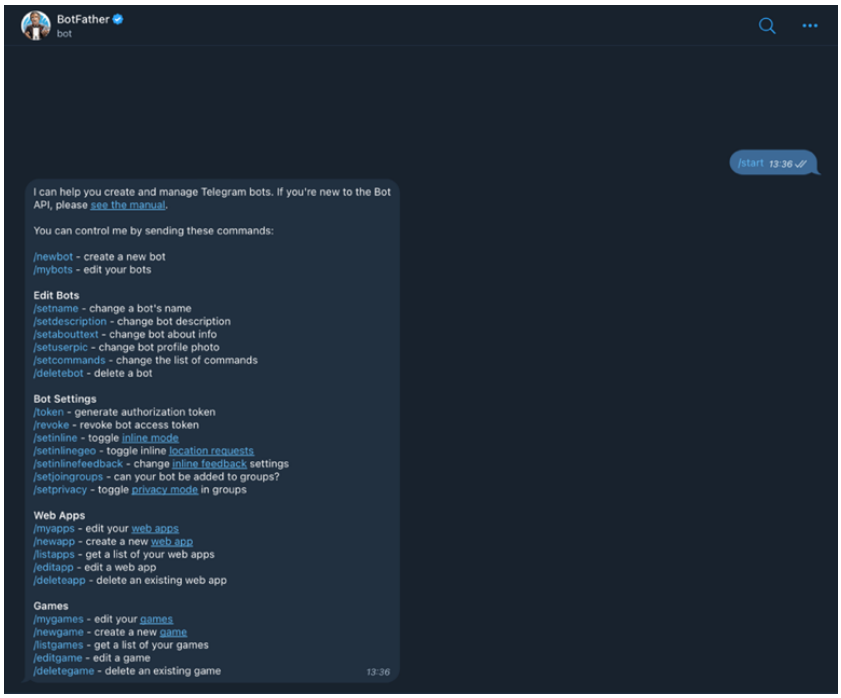

Рисунок 2.2 – Скріншот BotFather

При першому запуску бота користувачам пропонується пройти реєстрацію вводячи своє повне ім'я та номер телефону. Після реєстрації користувачам стають доступні функції, такі як записатись та переглянути записи (рис. 2.3).

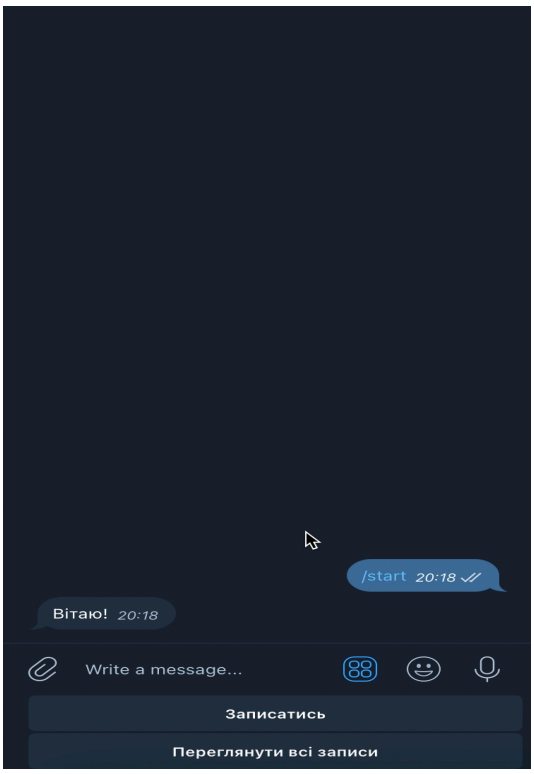

Рисунок 2.3 – Скріншот розробленого бота

Коли користувач натискає кнопку «Записатись» бот запрошує номер телефону, та пропонує послуги на які можна записатись. Коли користувач обрав послугу, бот перевіряє чи вільна послугу, якщо вільна то бот пропонує обрати вільну дату та час. Бот автоматично підтверджує запис, вказавши ім’я, номер телефону, послугу на яку записані, дату та час. Але якщо послуга не доступна бот повідомляє «Нажаль немає вільних дат на послугу» (рис. 2. 4, 2.5, 2.6).

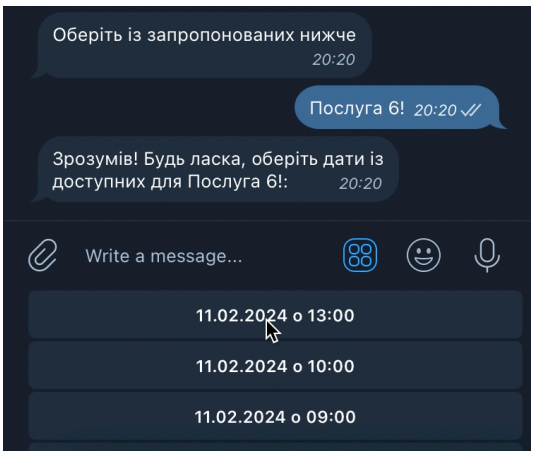

Рисунок 2.4 – Скріншот діалогу

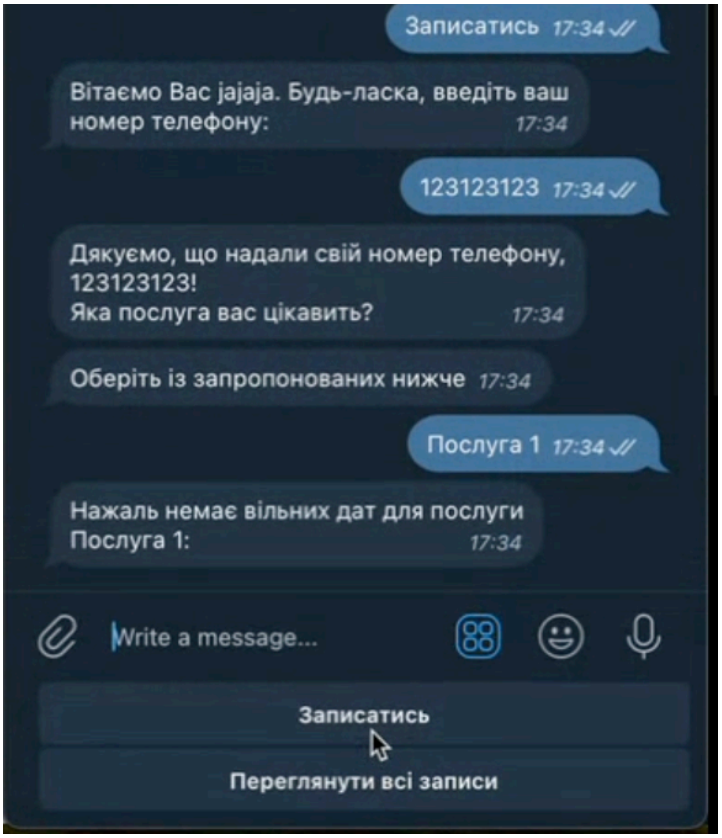

Рисунок 2.5 – Скріншот діалогу

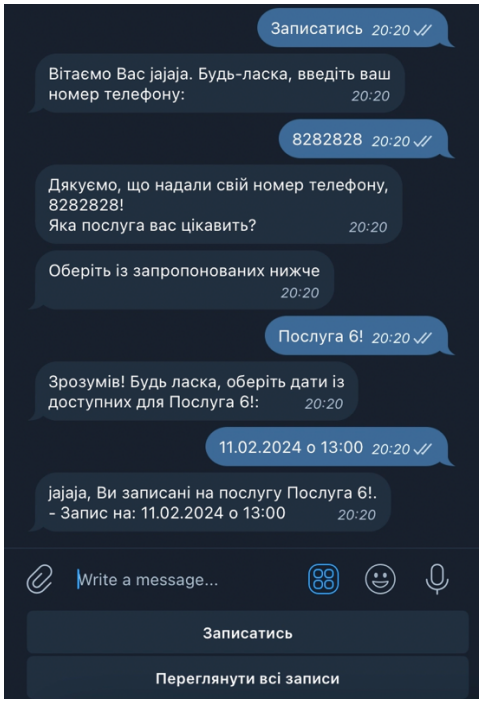

Рисунок. 2.6 – Скріншот діалогу

Коли користувач обирає кнопку «Переглянути всі записи», бот висилає користувачу повідомлення про підтверджений запис, а також пропонує підтвердити або скасувати запис на послугу (рис. 2.7).

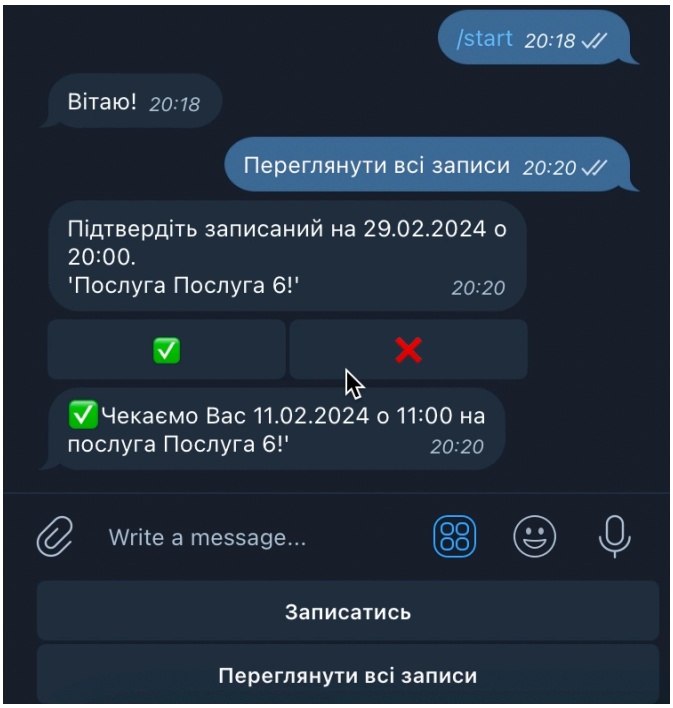

Рисунок 2.7 – Скріншот діалогу

Якщо почати писати боту він не зрозуміє, та буде пропонувати обрати з запропонованих варіантів які він надає (рис. 2.8).

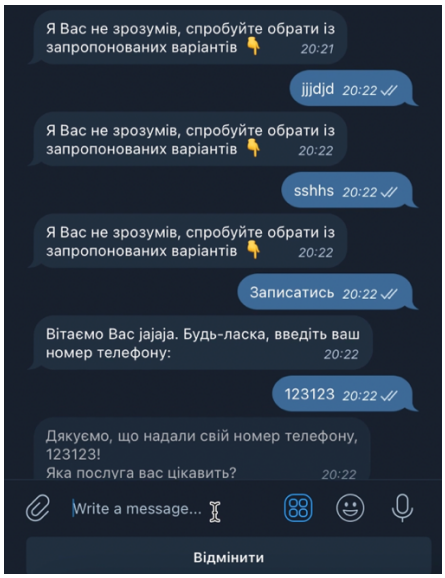

Рисунок 2.8 – Скріншот діалогу

Для зручності користування був розроблений веб-інтерфейс, він містить в собі сторінку з вільними датами, головну сторінку, послуги та клієнти (рис 2.9).

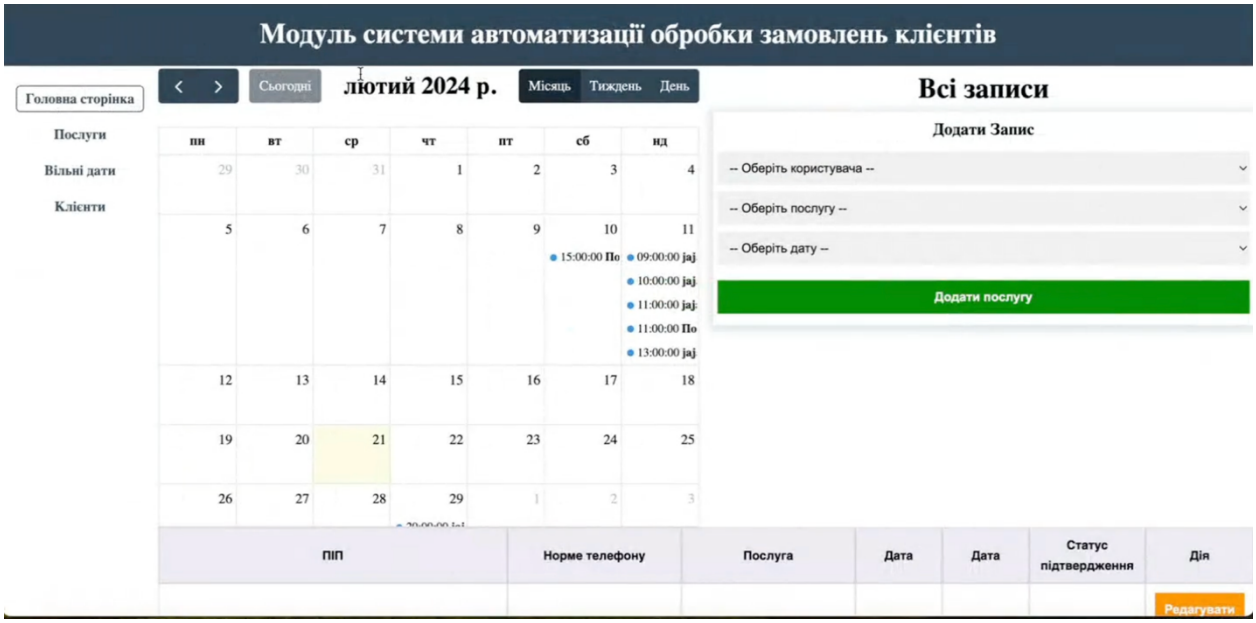

Рисунок 2.9 – Головна сторінка веб-інтерфейсу

На головній сторінці розміщений календар, де відображаються зарезервовані дати, можна переглянути інформацію про запис (рис. 2.10).

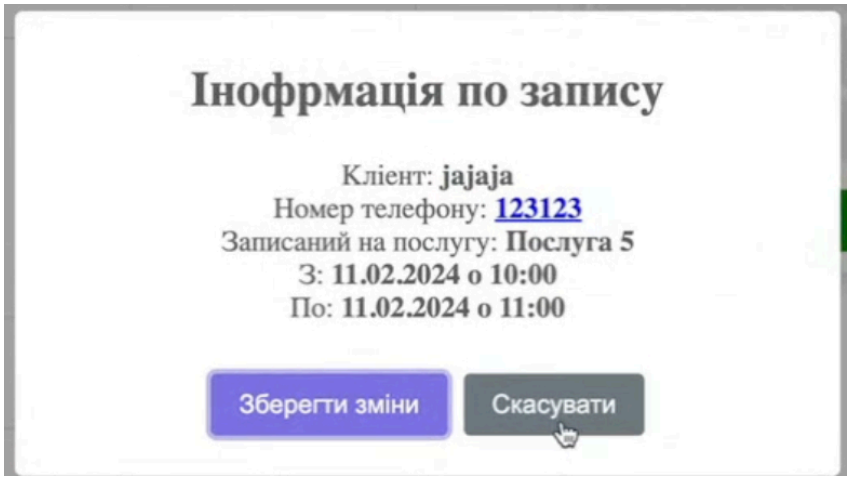

Рисунок 2.10 – Інформація про запис

Для зручності є таблиця в якій можна побачити підтверджені записи, якщо клієнт не підтвердив, можна зателефонувати та підтвердити, у клієнта в боті буде підтверджено запис автоматично, або ж скасувати запис (рис. 2.11).

Рисунок 2.11 – Таблиця

На сторінці Послуги, можна додавати послуги та кількість годин, редагувати або видаляти, і це все автоматично буде змінюватися в боті у користувача (рис.2.12).

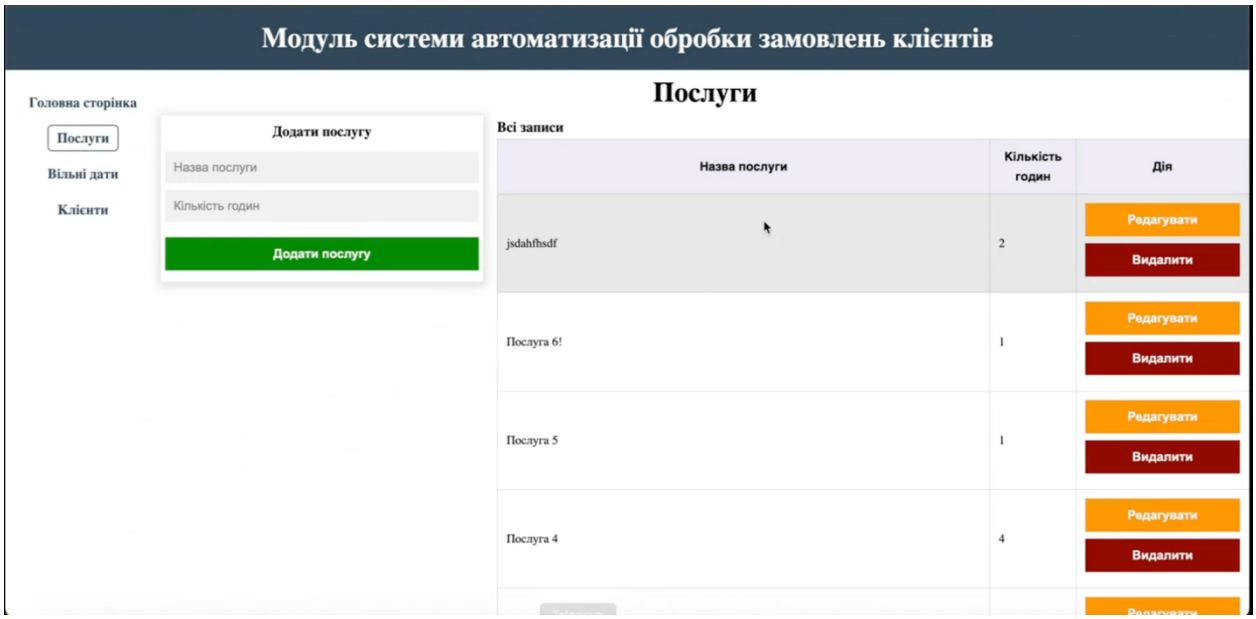

Рисунок 2.11 – Сторінка Послуги

На сторінці Вільні Дати, ми можемо додати вільну дату та час, а також побачити на яку дату та час є вже зарезервовані записи (рис. 2.13).

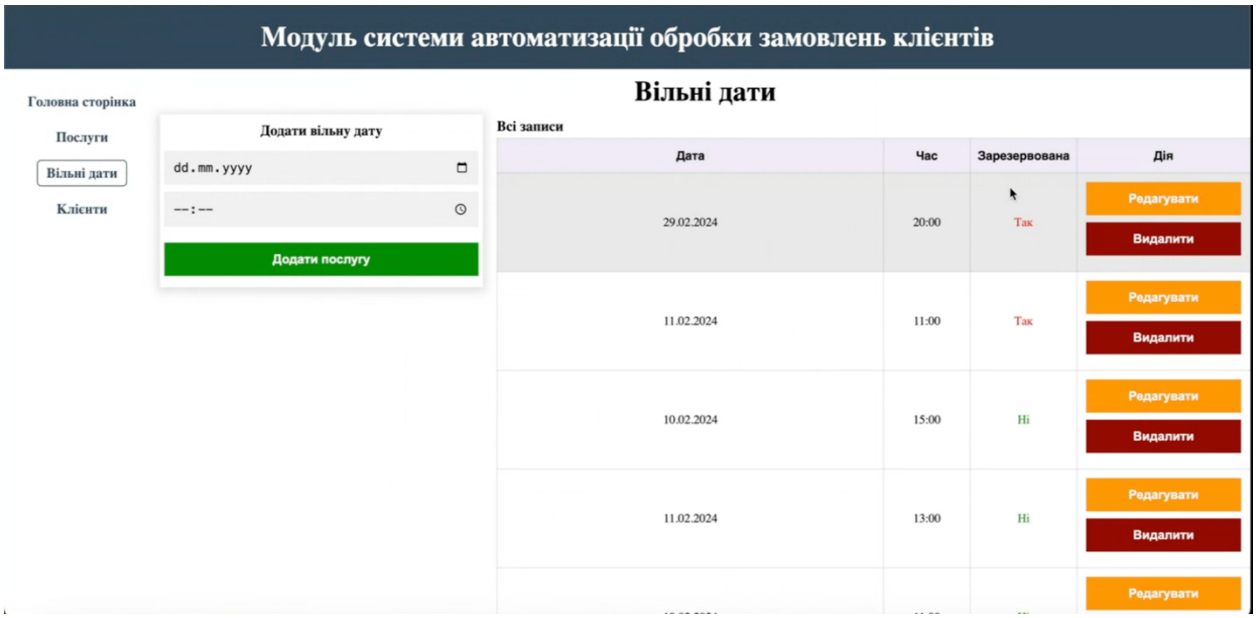

Рисунок 2.13 – Сторінка Вільні Дати

2.3 Оцінка очікуваного ефекту від впровадження модуля управління онлайн замовленнями

Автоматизація взаємодії між клієнтами та менеджерами може стати потужним інструментом для покращення ефективності роботи й прибутку вашого бізнесу. Однак, перед тим, як впроваджувати подібні рішення, важливо провести ретельний аналіз економічної доцільності, щоб чітко розуміти співвідношення очікуваних вигод та витрат.

Економічна ефективність – це ключовий показник, який визначає, наскільки вигідним буде впровадження автоматизації. Він розраховується як співвідношення отриманих результатів (економія коштів, покращення продуктивності, тощо) до витрат, понесених на розробку, впровадження та обслуговування системи [6].

Часто підприємці роблять помилку, поспішаючи з автоматизацією, не оцінивши всі її потенційні переваги. Це може призвести до неефективного використання ресурсів і розчарування результатами.

Автоматизація може призвести до значної економії коштів за рахунок зменшення потреби в людських ресурсах, зниження адміністративних витрат та оптимізації процесів.

Автоматизовані системи виконують завдання швидше та точніше за людей, що підвищує продуктивність і економить час.

Автоматизація може зробити взаємодію з клієнтами більш ефективною та зручною, що може призвести до збільшення їх задоволеності та лояльності.

Автоматизовані системи можуть допомогти вам краще зрозуміти потреби клієнтів і запропонувати їм більш персоналізовані продукти та послуги, що може призвести до збільшення продажів [1; 3; 34].

Важливо пам'ятати, що економічна ефективність автоматизації може варіюватися залежно від конкретного бізнесу та його потреб. Тому, перед тим, як приймати рішення про впровадження автоматизованої системи, важливо провести всебічний аналіз та чітко визначити свої цілі та очікування.

Впровадження модуля управління онлайн-замовленнями може мати значний позитивний вплив на ваш бізнес. Наприклад:

1. Покращення обслуговування клієнтів:

– Швидше виконання замовлень: Автоматизація процесів може призвести до значного скорочення часу, необхідного для обробки та виконання замовлень. Це може призвести до більш задоволених клієнтів і кращої репутації вашого бренду.

– Зменшення помилок: Модуль управління онлайн-замовленнями може допомогти зменшити кількість помилок, пов'язаних з людським фактором, таких як помилки введення даних або пропущені замовлення. Це може призвести до кращого досвіду для клієнтів і меншої кількості повернень або скарг.

– Доступність 24/7: Онлайн-замовлення можна розміщувати в будь-який час доби, що може бути зручніше для клієнтів, які живуть у різних часових поясах або мають щільний графік.

2. Збільшення продажів:

– Зручність для покупців: Простий і зручний процес онлайн-замовлення може призвести до збільшення кількості покупок.

– Можливості для перехресних продажів: Модуль управління онлайн-замовленнями може допомогти вам рекомендувати клієнтам інші продукти, які можуть їх зацікавити, що може призвести до збільшення середнього чека.

– Розширення ринку: Онлайн-продажі можуть допомогти вам охопити ширшу аудиторію, включаючи клієнтів, які не можуть відвідати ваш фізичний магазин.

3. Зниження витрат:

– Зменшення адміністративних витрат: Автоматизація процесів може призвести до значного скорочення часу та ресурсів, необхідних для управління онлайн-замовленнями. Це може звільнити ваших співробітників для роботи над іншими завданнями.

– Зменшення витрат на обробку замовлень: Модуль управління онлайн-замовленнями може допомогти зменшити витрати на обробку замовлень, такі як витрати на друк і поштові витрати.

– Зменшення крадіжок і шахрайства: Модуль управління онлайн-замовленнями може допомогти зменшити ризик крадіжок і шахрайства, забезпечуючи більш безпечний спосіб обробки платежів.

4. Покращення збору даних:

– Відстеження поведінки клієнтів: Модуль управління онлайн-замовленнями може збирати дані про поведінку клієнтів, такі як продукти, які вони переглядають, і товари, які вони купують. Ці дані можна використовувати для покращення вашого веб-сайту, маркетингових кампаній і продуктів.

– Аналіз тенденцій продажів: Модуль управління онлайн-замовленнями може допомогти вам відстежувати тенденції продажів і виявляти можливості для покращення вашого бізнесу.

Підходи до оцінки економічної ефективності автоматизації можуть варіюватися залежно від галузі, типу бізнесу та характеристик продукції. Це пов'язано з тим, що кожен бізнес має свої унікальні потреби та можливості, які потрібно враховувати при оцінці потенційної вигоди від автоматизації [6].

Серед доступних інструментів для автоматизації взаємодії з клієнтами, Telegram-боти вирізняються своєю доступністю та відкритістю. Їх безкоштовність та простота використання роблять їх привабливими для широкого кола користувачів, незалежно від розміру чи бюджету бізнесу. Незважаючи на те, що Telegram-боти пропонують безліч переваг, важливо порівняти їх з платними альтернативами на ринку, які пропонують подібні послуги за підпискою. Аналіз цін на платні сервіси чат-ботів показує, що вони можуть значно варіюватися залежно від обсягу запитів. Ціни можуть коливатися від 722 грн на місяць за менше 1000 підключень до 10785 грн від 500000 підключень [7].

При порівнянні цін на розробку чат-ботів (рис. 2.13), стає очевидно, що впровадження Telegram-бота може бути значно вигіднішим для малого та середнього бізнесу. Варто зазначити те, що Telegram-боти можна створити без значних фінансових витрат, що робить їх доступними для широкого кола підприємств. Розробка Telegram-бота не потребує глибоких знань програмування або спеціальних ресурсів з боку розробників. Це дозволяє швидко та без проблем впровадити чат-бота у ваш бізнес.

Щоб чітко оцінити економні переваги від використання Telegram-бота, важливо врахувати капітальні витрати, які б ми понесли, якби розробили сайт або окремий додаток, це витрати на програмне забезпечення та апаратне забезпечення, проектування, програмування, впровадження та налагодження [30].

Для розрахунку капітальних витрат на розробку Telegram-бота можна використовувати наступну формулу:

$$К = К_1 + К_2 + К_3 + К_4 + К_5 + К_6 \quad (2.1)$$

Де $К_1 - К_4$ – це заробітна плата програмістів, вона включає в себе оплату праці фахівців, які розроблять код чат-бота або веб-сайту, а також оплату праці фахівців з тестування та налагодження системи.

$К_4$– це витрати на налаштування та тестування, включає в себе оплату праці фахівців, які налаштують чат-бота або веб-сайт відповідно до ваших потреб, а також оплату праці фахівців, які протестують систему та усунуть будь-які помилки.

$К_5$ – це витрати на програмне забезпечення, включає в себе вартість ліцензій на програмне забезпечення, яке використовується для розробки та експлуатації чат-бота або веб-сайту.

$К_6$ – це витрати на обладнання, включає в себе вартість серверів, хостингу та інших компонентів, необхідних для роботи чат-бота або веб-сайту.

При оцінці загальних витрат на розробку та впровадження програмного забезпечення важливо ретельно врахувати не лише витрати на оплату праці програмістів, але й інші супутні витрати, пов'язані з необхідними ресурсами. Для розробки чат-бота потребуватиме 70 годин робочого часу. Ця оцінка ґрунтується на низці факторів, таких як: складність чат-бота, досвід розробників, наявність інструментів та ресурсів. Середньо місячна заробітна плата програміста складає 100 230 грн, беручи 40 годинний робочий тиждень, годинна ставка становить 626,44 грн./год. Врахуємо ще оплату хостингу, вона становить 2406 грн. на рік за тарифом Nanode 1 GB [14; 16].

Підсумовуючи, загальні капітальні витрати на розробку та впровадження чат-бота можна розрахувати наступним чином:

$$К = 70 \cdot 626,44 + 0 + 2406 = 46\ 257 \text{ (грн)}$$

Платні сервіси Telegram-ботів пропонують широкий спектр функцій, які можуть бути корисними для бізнесу. Однак, важливо тверезо оцінити, чи дійсно всі ці функції необхідні для вашого проекту. Існує ймовірність, що деякі з них виявляться зайвими, що призведе до невиправданих витрат. В таких випадках, розробка власного Telegram-бота може стати більш вигідним та раціональним рішенням. Це дозволить суттєво скоротити витрати, платити лише за ті функції, які дійсно будете використовувати. Отримати необхідний функціонал завдяки якому можна розробити бота, який буде відповідати вашим конкретним потребам та завданням. Уникнути переплат, не доведеться платити за функції, які не потрібні. Завдяки такому підходу можна заощадити до 46 257 грн. Це значна сума, яку можна використовувати для розвитку вашого бізнесу іншими способами. Отже, розробка власного Telegram-бота може бути ефективним та економічно вигідним рішенням для вашого бізнесу.

ВИСНОВОК

Розробка Telegram-бота для автоматизації обробки замовлень клієнтів СМБ може стати вигідним інструментом для покращення роботи та обслуговування клієнтів для багатьох компаній. Завдяки автоматизації рутинних завдань, пов'язаних з обробкою замовлень, Telegram-бот може допомогти:

- Зменшити час: Швидка обробка замовлень призводить до задоволеності клієнтів.
- Знизити витрати: Економія коштів на персоналі та обробці замовлень може бути інвестована в інші сфери розвитку бізнесу.
- Підвищити задоволеність клієнтів: Швидке, якісне та цілодобове обслуговування веде до кращого досвіду для клієнтів, що сприяє їх лояльності та повторним записам.
- Покращити імідж: Позитивні відгуки клієнтів про роботу бота можуть покращити репутацію СМБ та підвищити його конкурентоспроможність.

В ході роботи було створено Telegram-бот, який володіє значним потенціалом для оптимізації роботи СМБ. Завдяки цьому клієнти матимуть можливість легко реєструватися та записуватися на послуги через Telegram-бот, а адміністратори зможуть ефективно керувати цими замовленнями через адміністративну панель. Використання сучасних технологій, таких як React.js, Flask, aiogram та інші, дозволить забезпечити швидку та надійну роботу системи. Впровадження Telegram-бота може принести значну користь СМБ, оптимізуючи їхні процеси, економлячи кошти та стимулюючи зростання бізнесу.

Важливо зазначити, що успішність Telegram-бота також залежить від його постійного вдосконалення та адаптації до мінливих потреб клієнтів та ринку. Регулярний збір відгуків та аналіз даних про використання бота допоможуть його розробникам вносити необхідні зміни та покращення, гарантуючи його довгострокову ефективність.

У результаті, розробка цього модуля сприятиме покращенню обслуговування клієнтів, підвищенню ефективності бізнесу та забезпеченню конкурентоспроможності компанії на ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Автоматизація бізнесу: як вона допомагає заробляти більше. *finance.ua*. [Електронний ресурс]. Доступно: https://finance.ua/ua/goodtoknow/avtomatyzacia-biznesu

2. Все про чат-боти: типи і приклади, якому бізнесу підійде, список конструкторів для створення. *Webpromo*. [Електронний ресурс]. Доступно: https://web-promo.ua/ua/blog/vse-o-chat-botah-tipy-i-primery-kakomu-biznesu-podojdet-spisok-konstruktorov-dlya-sozdaniya/#types

3. Грек К. А. Автоматизований клієнтський сервіс трекінгу поштових відправлень : робота на здобуття кваліфікаційного ступеня магістра за спеціальністю 051 - економіка / К. А. Грек; наук. кер. К. Г. Гриценко. – Суми : Сумський державний університет, 2022. – 56 с.

4. Головна - Степанівська загальноосвітня школа I-III ступенів Степанівської селищної ради Сумського району Сумської області. [Електронний ресурс]. Доступно: http://stepanivka-school1.edukit.sumy.ua/Files/downloads/Book-Python.pdf

5. Де використовується Python і чому вам потрібно знати цю мову - Genius.Space. *Genius.Space*. [Електронний ресурс]. Доступно: https://genius.space/lab/de-vikoristovuyetsya-python-i-chomu-vam-potribno-znati-tsyu-movu/

6. Економічна ефективність. *Енциклопедія Сучасної України*. [Електронний ресурс]. Доступно: https://esu.com.ua/article-18769

7. Розробка та створення чат бота telegram, viber, fb, instagram – компанія Gerabot [Електронний ресурс]. Доступно: https://gerabot.com/

8. Оцініть продукти Creatio для продажів, маркетингу, сервису та управління бізнес-процесами | Creatio. *No-Code Platform to automate workflows and CRM | Creatio*. [Електронний ресурс]. Доступно: https://www.creatio.com/ua/trial/creatio?activity=adwords_brand_ua&utm_term=sales%20creatio

9. Що таке Magento? Все, що вам потрібно знати. *403 Forbidden*. [Електронний ресурс]. Доступно: https://magefan.com/ua/blog/shcho-take-magento

10. Що таке Python і де він використовується. [Електронний ресурс]. Доступно: https://dan-it.com.ua/uk/blog/python-chto-jeto-za-jazyk-programmirovanija-i-gde-ego-ispolzujut/

11. Як створити онлайн-магазин на Shopify і почати продавати товари у 175 країнах – інструкція – AIN.UA. *AIN.UA*. [Електронний ресурс]. Доступно: https://ain.ua/2021/08/23/yak-stvoryty-onlajn-magazyn-na-shopify-i-pochaty-prodavaty-tovary-u-175-krayinah-instrukcziya/

12. 5 types of chatbot and how to choose the right one. *IBM Blog*. [Електронний ресурс]. Доступно: https://www.ibm.com/blog/chatbot-types/

13. aiogram 3.7.0 documentation. *aiogram 3.7.0 documentation*. [Електронний ресурс]. Доступно: https://docs.aiogram.dev/uk-ua/latest/

14. Cloud Computing Services [Електронний ресурс]. Доступно: https://www.linode.com/pricing/

15. dbdiagram.io - Database Relationship Diagrams Design Tool. *dbdiagram.io - Database Relationship Diagrams Design Tool*. [Електронний ресурс]. Доступно: https://dbdiagram.io/d/664f59e9f84ecd1d22f9ff6a

16. DOU: Спільнота програмістів [Електронний ресурс]. Доступно: https://jobs.dou.ua/salaries/?period=2023-12&position=Middle%20SE&technology=Python

17. FullCalendar - JavaScript Event Calendar. *FullCalendar - JavaScript Event Calendar*. [Електронний ресурс]. Доступно: https://fullcalendar.io

18. Getting Started | Axios Docs. *Axios*. [Електронний ресурс]. Доступно: https://axios-http.com/docs/intro

19. How to select the best chatbot platforms for your business in 2024 - reviews, features, pricing, comparison - PAT RESEARCH: B2B reviews, buying guides & best practices. *PAT RESEARCH: B2B Reviews, Buying Guides & Best*

Practices. [Електронний ресурс]. Доступно: https://www.predictiveanalyticstoday.com/what-is-chatbot-platform/

20. Iiba. Business analysis blog | why defining the business need is critical | IIBA. *Business Analysis | The Global Standard | IIBA®*. [Електронний ресурс]. Доступно: https://www.iiba.org/business-analysis-blogs/why-defining-the-business-need-is-critical/?creative=&keyword=&matchtype=&network=x&device=c&gad_source=1&gclid=Cj0KCQjw2a6wBhCVARIsABPeH1umBCWeI8CJJLF8EaJPLBLLVRM5aHbaszo8Nd8i4hEqEqaxUwrK_sQaAvtaEALw_wcB

21. Node.js – Download Node.js®. *Node.js – Run JavaScript Everywhere*. [Електронний ресурс]. Доступно: https://nodejs.org/en/download/prebuilt-installer

22. Oracle Ukraine | Cloud Applications and Cloud Platform. *Oracle | Cloud Applications and Cloud Platform*. [Електронний ресурс]. Доступно: https://www.oracle.com/ua/

23. Python. [Електронний ресурс]. Доступно: https://pythonguide.rozh2sch.org.ua

24. Projector Creative & Tech Institute. React: Що таке React? Як почати вивчати Реакт? Основні навички. *CASES*. [Електронний ресурс]. Доступно: https://cases.media/article/sho-take-react-js-yak-pochati-vivchati-reakt-navichki-dlya-react-developer

25. Quick Start – React. *React*. [Електронний ресурс]. Доступно: https://react.dev/learn

26. Rud A. Що таке Node.js та для чого він потрібен? | Блог HyperHost.UA. *Український хостинг провайдер HyperHost. Купити хостинг для сайту*. [Електронний ресурс]. Доступно: https://hyperhost.ua/info/uk/shho-take-nodejs-ta-dlya-cogo-vin-potriben

27. SAP Business One [Електронний ресурс]. Доступно: https://www.sap.com/products/erp/business-one.html

28. Start and grow your e-commerce business - 3-Day Free Trial. *Shopify*. [Електронний ресурс]. Доступно: https://www.shopify.com

29. Sign in. *Sign in*. [Електронний ресурс]. Доступно: https://account.magento.com/customer/account/login

30. SumDU Repository: Home. [Електронний ресурс]. Доступно: https://essuir.sumdu.edu.ua/bitstream-download/123456789/68533/1/Zubkov_mah_rob.pdf;jsessionid=36D6346AE0F5E3AD1FA25A4293AFC975

31. SweetAlert2. *SweetAlert2 - a beautiful, responsive, customizable and accessible (WAI-ARIA) replacement for JavaScript's popup boxes*. [Електронний ресурс]. Доступно: https://sweetalert2.github.io

32. SQLite home page. *SQLite Home Page*. [Електронний ресурс]. Доступно: https://www.sqlite.org

33. Welcome to Flask – Flask Documentation (3.0.x). *Welcome to Flask – Flask Documentation (3.0.x)*. [Електронний ресурс]. Доступно: https://flask.palletsprojects.com/en/3.0.x/

34. What is Business Process Automation: Definition, examples, and software | Snov.io. *Sales automation & acceleration at scale | Snov.io*. [Електронний ресурс]. Доступно: https://snov.io/glossary/business-process-automation/

35. UX дослідження: методи аналізу та визначення потреб користувачів. *Корисні матеріали: Статті та новини IT-індустрії | Комп'ютерна школа Hillel*. [Електронний ресурс]. Доступно: https://blog.ithillel.ua/articles/best-practices-for-determining-user-needs

ДОДАТКИ

ДОДАТОК А

SUMMARY

Putinцева A. V. Development of a Module of the Customer Order Processing Automation System for Medium and Small Businesses. Bachelor's qualification work. Sumy State University, Sumy, 2024.

This thesis presents the development of a Telegram bot aimed at automating the order processing process for customers. An analysis of existing online order management systems was conducted. User and business needs were identified, and a technical specification for the development of an online order management module was developed, including the choice of a technology stack, functional capabilities, and architecture. The effectiveness of the module and its impact on the operational activities of SMEs were evaluated.

Keywords automation, Telegram bot, Order management, Flask, React.js, SQLite, Aiogram, FullCalendar.io, Python-dotenv, Python, Node.js.

АНОТАЦІЯ

Путінцева А. В. Розробка модуля системи автоматизації обробки замовлень клієнтів для середнього та малого бізнесу. Кваліфікаційна робота бакалавра. Сумський державний університет, м. Суми, 2024 р.

У роботі було розроблено Telegram-бот, призначеного для автоматизації процесу обробки замовлень клієнтів. Було проведено аналіз існуючих систем управління онлайн замовленнями. Були визначені потреби користувачів та бізнесу, розроблене технічне завдання до розробки модуля управління онлайн замовленнями, включаючи вибір технологічного стека, функціональні можливості та архітектуру. Визначили оцінку ефективності модуля та його вплив на операційну діяльність СМБ.

Ключові слова: автоматизація, Telegram-бот, управління замовленнями, записи, послуги, чат-бот, Flask, React.js, SQLite, aiogram, FullCalendar.io, python-dotenv Python, Node.js.

ДОДАТОК Б

Фрагмент коду файлу app.py

```
from flask import Flask, request, jsonify
from flask_cors import CORS
from dotenv import load_dotenv
import os
from database import get_all_appointments, add_user_to_db, add_service_to_db,
\
    add_available_dates_to_db, add_appointment_to_db, get_user_info_by_id,
get_service_info_by_id, get_date_info_by_id, \
    get_all_users_from_db, get_all_available_dates_from_db,
get_all_services_from_db, delete_appointment_from_db, \
    update_appointment_in_db, update_available_date_in_db,
delete_available_date_from_db, update_user_in_db, \
    delete_user_from_db, update_service_in_db, delete_service_from_db,
get_available_dates_for_service, \
    set_status_appointment_in_db, get_user_by_id

from datetime import datetime, timedelta

load_dotenv()

app = Flask(__name__)
CORS(app)


@app.route('/api/appointments', methods=['GET'])
def get_appointments():
    appointments_list = []

    user_id = request.args.get('user_id')

    if user_id:
        appointments = get_user_by_id(user_id)
    else:
        appointments = get_all_appointments()
    for appointment in appointments:
        user_info = get_user_info_by_id(appointment['user_id'])
        service_info = get_service_info_by_id(appointment['service_id'])
        date_info = get_date_info_by_id(appointment['date_id'])

        if user_info and service_info and date_info:
            start_datetime =
datetime.strptime(f"{date_info['available_date']}
{date_info['available_time']}",
                                                "%Y-%m-%d %H:%M")
            end_datetime = start_datetime +
timedelta(hours=service_info['duration_hours'])

            appointment_dict = {
                'id': appointment['id'],
                'user_id': appointment['user_id'],
                'service_id': appointment['service_id'],
                'date_id': appointment['date_id'],
                'title': user_info['name'],
                'phone': user_info['phone'],
                'service': service_info['service_name'],
                'start': start_datetime.isoformat(),
                'end': end_datetime.isoformat(),
                'status': appointment['status'],
                'telegram_id': user_info['telegram_id'],
            }
```

Продовження додатка Б

```
            appointments_list.append(appointment_dict)
    appointments_list.reverse()
    return jsonify({'appointments': appointments_list})


# Route to add a new appointment
@app.route('/api/appointments', methods=['POST'])
def add_appointment():
    try:
        data = request.get_json()

        user_id = data.get('user_id')
        service_id = data.get('service_id')
        date_id = data.get('date_id')

        if not user_id or not service_id or not date_id:
            return jsonify({'error': 'Invalid input. user_id, service_id, and
date_id are required.'}), 400

        success = add_appointment_to_db(user_id, service_id, date_id)

        if success:
            return jsonify({'message': 'Appointment added successfully!'})
        else:
            return jsonify({'error': 'Failed to add appointment to the
database.'}), 500

    except Exception as e:
        return jsonify({'error': str(e)}), 500


@app.route('/api/appointments/<int:appointment_id>', methods=['PUT'])
def update_appointment(appointment_id):
    try:
        data = request.get_json()

        user_id = data.get('user_id')
        service_id = data.get('service_id')
        date_id = data.get('date_id')

        if not user_id or not service_id or not date_id:
            return jsonify({'error': 'Invalid input. user_id, service_id, and
date_id are required.'}), 400

        success = update_appointment_in_db(appointment_id, user_id,
service_id, date_id)

        if success:
            return jsonify({'message': 'Appointment updated successfully!'})
        else:
            return jsonify({'error': 'Failed to update appointment in the
database.'}), 500

    except Exception as e:
        return jsonify({'error': str(e)}), 500


@app.route('/api/appointments/<int:appointment_id>', methods=['POST'])
def update_appointment_status(appointment_id):
```

Продовження додатка Б

```
    try:
        success = set_status_appointme
nt_in_db(appointment_id)

        if success:
            return jsonify({'message': 'Appointment status updated
successfully!'})
        else:
            return jsonify({'error': 'Failed to update appointment in the
database.'}), 500

    except Exception as e:
        return jsonify({'error': str(e)}), 500


@app.route('/api/appointments/<int:appointment_id>', methods=['DELETE'])
def delete_appointment(appointment_id):
    success = delete_appointment_from_db(appointment_id)
    if success:
        return jsonify({'message': 'Appointment deleted successfully!'})
    else:
        return jsonify({'error': 'Failed to delete appointment from the
database.'}), 500


# Route to get all available dates
@app.route('/api/available_dates', methods=['GET'])
def get_available_dates():
    try:
        service_id = request.args.get('service_id')

        if service_id:
            available_dates = get_available_dates_for_service(service_id)
        else:
            available_dates_list = get_all_available_dates_from_db()
            available_dates = [
                {'id': date['id'], 'available_date': date['available_date'],
'available_time': date['available_time'],
                 'is_appointments': date['is_appointments']} for date in
available_dates_list]
        available_dates.reverse()
        return jsonify({'available_dates': available_dates})
    except Exception as e:
        return jsonify({'error': str(e)}), 500


# Route to add available dates
@app.route('/api/available_dates', methods=['POST'])
def add_available_dates():
    try:
        data = request.get_json()
        date = data.get('available_date')
        time = data.get('available_time')
        if not date or not time:
            return jsonify({'error': 'Invalid input. Both available_date and
available_time are required.'}), 400
        success = add_available_dates_to_db(date, time)
        if success:
            return jsonify({'message': 'Available dates added
successfully!'})
```

Продовження додатка Б

```
        else:
            return jsonify({'error': 'Failed to add available dates to the
database.'}), 500
    except Exception as e:

return jsonify({'error': str(e)}), 500


@app.route('/api/available_dates/<int:date_id>', methods=['PUT'])
def update_available_date(date_id):
    try:
        data = request.get_json()
        date = data.get('available_date')
        time = data.get('available_time')

        if not date or not time:
            return jsonify({'error': 'Invalid input. Both available_date and
available_time are required.'}), 400

        success = update_available_date_in_db(date_id, date, time)

        if success:
            return jsonify({'message': 'Available date updated
successfully!'})
        else:
            return jsonify({'error': 'Failed to update available date in the
database.'}), 500

    except Exception as e:
        return jsonify({'error': str(e)}), 500


@app.route('/api/available_dates/<int:date_id>', methods=['DELETE'])
def delete_available_date(date_id):
    success = delete_available_date_from_db(date_id)
    if success:
        return jsonify({'message': 'Available date deleted successfully!'})
    else:
        return jsonify({'error': 'Failed to delete available date from the
database.'}), 500


# Route to get all users
@app.route('/api/users', methods=['GET'])
def get_all_users():
    try:
        users = get_all_users_from_db()
        users_list = [
            {'id': user['id'], 'telegram_id': user['telegram_id'], 'name':
user['name'], 'phone': user['phone']} for
            user in users]
        users_list.reverse()
        return jsonify({'users': users_list})
    except Exception as e:
        return jsonify({'error': str(e)}), 500


# Route to add a new user
@app.route('/api/users', methods=['POST'])
def add_user():
```

Продовження додатка Б

```
    try:
        data = request.get_json()
        telegram_id = data.get('telegram_id')
        name = data.get('name')
        phone = data.get('phone')
        if not telegram_id or not name or not phone:
            return jsonify({'error': 'Invalid input. Both telegram_id, name
and phone are required.'}), 400
        success = add_user_to_db(telegram_id, name, phone)
        if success:
            return jsonify({'message': 'User added successfully!'})
        else:
            return jsonify({'error': 'Failed to add user to the database.'}),
500
    except Exception as e:
        return jsonify({'error': str(e)}), 500


@app.route('/api/users/<int:user_id>', methods=['PUT'])
def update_user(user_id):
    try:
        data = request.get_json()
        telegram_id = data.get('telegram_id')
        name = data.get('name')
        phone = data.get('phone')

        if not telegram_id or not name or not phone:
            return jsonify({'error': 'Invalid input. Both telegram_id, name
and phone are required.'}), 400

        success = update_user_in_db(user_id, telegram_id, name, phone)

        if success:
            return jsonify({'message': 'User updated successfully!'})
        else:
            return jsonify({'error': 'Failed to update user in the
database.'}), 500

    except Exception as e:
        return jsonify({'error': str(e)}), 500


@app.route('/api/users/<int:user_id>', methods=['DELETE'])
def delete_user(user_id):
    success = delete_user_from_db(user_id)
    if success:
        return jsonify({'message': 'User deleted successfully!'})
    else:
        return jsonify({'error': 'Failed to delete user from the
database.'}), 500


# Route to get all services
@app.route('/api/services', methods=['GET'])
def get_all_services():
    try:
        services = get_all_services_from_db()
        services_list = [
            {'id': service['id'], 'service_name': service['service_name'],
```

Продовження додатка Б

```
'duration_hours': service['duration_hours']}
                for service in services]
            services_list.reverse()
            return jsonify({'services': services_list})
        except Exception as e:
            return jsonify({'error': str(e)}), 500


# Route to add a new service
@app.route('/api/services', methods=['POST'])
def add_service():
    try:
        data = request.get_json()

        service_name = data.get('service_name')
        duration_hours = data.get('duration_hours')

        if not service_name or not duration_hours:
            return jsonify({'error': 'Invalid input. Both service_name and
duration_hours are required.'}), 400

        success = add_service_to_db(service_name, duration_hours)

        if success:
            return jsonify({'message': 'Service added successfully!'})
        else:
            return jsonify({'error': 'Failed to add service to the
database.'}), 500

    except Exception as e:
        return jsonify({'error': str(e)}), 500


@app.route('/api/services/<int:service_id>', methods=['PUT'])
def update_service(service_id):
    try:
        data = request.get_json()
        service_name = data.get('service_name')
        duration_hours = data.get('duration_hours')

        if not service_name or not duration_hours:
            return jsonify({'error': 'Invalid input. Both service_name and
duration_hours are required.'}), 400

        success = update_service_in_db(service_id, service_name,
duration_hours)

        if success:
            return jsonify({'message': 'Service updated successfully!'})
        else:
            return jsonify({'error': 'Failed to update service in the
database.'}), 500

    except Exception as e:
        return jsonify({'error': str(e)}), 500


@app.route('/api/services/<int:service_id>', methods=['DELETE'])
def delete_service(service_id):
    success = delete_service_from_db(service_id)
    if success:
```

Продовження додатка Б

```
        return jsonify({'message': 'Service deleted successfully!'})
    else:
        return jsonify({'error': 'Failed to delete service from the
database.'}), 500


if __name__ == '__main__':

    app.run(debug=True, port=os.getenv('PORT'))
```

Фрагмент коду bot.py

```
import os
import asyncio
import logging
import sys

from datetime import datetime, timedelta
import requests

from dotenv import load_dotenv

from aiogram.filters.callback_data import CallbackData
from aiogram import Bot, Dispatcher, F, Router, html
from aiogram.enums import ParseMode
from aiogram.filters import CommandStart
from aiogram.fsm.context import FSMContext
from aiogram.fsm.state import State, StatesGroup
from aiogram.utils.keyboard import InlineKeyboardBuilder,
ReplyKeyboardBuilder
from aiogram.types import (
    KeyboardButton,
    Message,
    ReplyKeyboardMarkup,
    ReplyKeyboardRemove,
    CallbackQuery
)

load_dotenv()

TOKEN = os.getenv('TG_TOKEN')
API_URL = os.getenv('API_URL')
logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO)

router = Router()
bot = Bot(token=TOKEN, parse_mode=ParseMode.HTML)


class Form(StatesGroup):
    name = State()
    phone = State()
    service = State()
    available_dates = State()


class AppointmentsCallbackConfirmation(CallbackData, prefix="appointment"):
    action: str
```

Продовження додатка Б

```
    id: int


def get_keyboard_appointments(appointment_id):
    builder = InlineKeyboardBuilder()
    builder.button(
        text="✅",
callback_data=AppointmentsCallbackConfirmation(action="confirm",
id=appointment_id)
    )
    builder.button(
        text="❌",
callback_data=AppointmentsCallbackConfirmation(action="cancel",
id=appointment_id)
    )
    builder.adjust(2)
    return builder.as_markup()


def get_data(attribute):
    url = f"{API_URL}/{attribute}"
    response = requests.get(url)
    data = response.json()
    return data


def get_service_id_by_name(attribute, key, value):
    url = f"{API_URL}/{attribute}"
    response = requests.get(url)
    data = response.json()
    items = data[attribute]
    for item in items:
        if item[key] == value:
            return item["id"]
    return None


def get_id_by_date_and_time(attribute, date_time):
    url = f"{API_URL}/{attribute}"
    original_date_string, time = date_time.split(' o ')
    original_date = datetime.strptime(original_date_string, "%d.%m.%Y")
    formatted_date = original_date.strftime("%Y-%m-%d")
    response = requests.get(url)
    data = response.json()
    for available_date in data["available_dates"]:
        if available_date["available_date"] == formatted_date and
available_date["available_time"] == time:
            return available_date["id"]
    return None


def get_user_by_tg_id(attribute, tg_id):
    url = f"{API_URL}/{attribute}"
    response = requests.get(url)
    data = response.json()
    for item in data[attribute]:
        if item["telegram_id"] == tg_id:
```

Продовження додатка Б

```
            return item
    return None


def get_item_by_id(attribute, key, value):
    url = f"{API_URL}/{attribute}"
    response = requests.get(url)
    data = response.json()
    items = data[attribute]
    for item in items:
        if item[key] == value:
            return item
    return None


def format_date(original_date_string):
    try:
        original_date = datetime.strptime(original_date_string, "%Y-%m-%d о
%H:%M")
        formatted_date = original_date.strftime("%d.%m.%Y о %H:%M")
        return formatted_date
    except ValueError:
        print("Error: Incorrect date format. Please provide a date string in
the format 'YYYY-MM-DD о HH:MM'")
        return None


def get_keyboard_start():
    builder = ReplyKeyboardBuilder()
    builder.add(KeyboardButton(text="Записатись"))
    builder.add(KeyboardButton(text="Переглянути всі записи"))
    builder.adjust(1)
    return builder.as_markup()


def get_keyboard_cancel():
    builder = ReplyKeyboardBuilder()
    builder.add(KeyboardButton(text="Відмінити"))
    builder.adjust(1)
    return builder.as_markup()


@router.message(CommandStart())
async def command_start_handler(message: Message) -> None:
    await message.answer("Вітаю!", reply_markup=get_keyboard_start())


@router.message(F.text.lower() == "записатись")
async def progress_start(message: Message, state: FSMContext) -> None:
    current_user = get_user_by_tg_id("users", message.chat.id)
    if current_user is None:
        await state.set_state(Form.name)
        await message.answer(
            "Введіть Ваше ПІП:",
            reply_markup=get_keyboard_cancel(),
        )
    else:
        await message.answer(
            f"Вітаємо Вас {current_user['name']}. Будь-ласка, введіть ваш
номер телефону: ",
            reply_markup=get_keyboard_cancel(),
```

Продовження додатка Б

```
        )
        await state.set_state(Form.phone)
        await state.update_data(user=message.chat.id)


@router.message(F.text.lower() == "переглянути всі записи")
async def show_user_appointments(message: Message) -> None:
    url = f"{API_URL}/appointments?user_id={message.chat.id}"
    data = requests.get(url).json()
    for appointment in data['appointments']:
        appointment_start_message = datetime.strptime(appointment['start'],
"%Y-%m-%dT%H:%M:%S").strftime(
            "%d.%m.%Y о %H:%M")
        if appointment['status'] == 0:
            message_text = f"Підтвердіть записаний на
{appointment_start_message}.\n'Послуга {appointment['service']}'"
            await bot.send_message(appointment["telegram_id"], message_text,

reply_markup=get_keyboard_appointments(appointment['id']))
            get_keyboard_start()
        else:
            message_text = f"✅Чекаємо Вас {appointment_start_message} на
послуга {appointment['service']}'"
            await bot.send_message(appointment["telegram_id"], message_text,
                                   reply_markup=get_keyboard_start())


@router.message(F.text.lower() == "відмінити")
async def progress_cancel(message: Message, state: FSMContext) -> None:
    await message.answer("Відміна запису!",
reply_markup=get_keyboard_start())


@router.message(Form.name)
async def process_name(message: Message, state: FSMContext) -> None:
    await state.update_data(name=message.text)
    await message.answer(
        f"Приємно познайомитися, {html.quote(message.text)}!\nБудь-ласка,
введіть ваш номер телефону:",
        reply_markup=get_keyboard_cancel(),
    )
    await state.set_state(Form.phone)


@router.message(Form.phone)
async def process_phone(message: Message, state: FSMContext) -> None:
    current_user = get_user_by_tg_id("users", message.chat.id)
    data = await state.update_data(phone=message.text)
    if current_user is None:
        url = f"{API_URL}/users"
        requests.post(url, json={
            "telegram_id": message.chat.id,
            "name": data['name'],
            "phone": data['phone']
        })
    else:
        url = f"{API_URL}/users/{current_user['id']}"
        requests.put(url, json={
            "telegram_id": message.chat.id,
            "name": current_user['name'],
            "phone": data['phone']
```

Продовження додатка Б

```
        })
    await state.clear()
    await state.update_data(user=message.chat.id)
    services_data = get_data("services")['services']
    keyboard_buttons = []
    for item in services_data:
        service_name = item["service_name"]
        kb = [KeyboardButton(text=service_name)]
        keyboard_buttons.append(kb)
    c_bk = [KeyboardButton(text="Відмінити")]
    keyboard_buttons.append(c_bk)
    keyboard = ReplyKeyboardMarkup(
        keyboard=keyboard_buttons,
        resize_keyboard=True
    )
    await state.set_state(Form.service)
    await message.answer(
        f"Дякуємо, що надали свій номер телефону,
{html.quote(message.text)}!\nЯка послуга вас цікавить?",
        reply_markup=get_keyboard_cancel(),
    )
    await message.answer(
        f"Оберіть із запропонованих нижче",
        reply_markup=keyboard,
    )


@router.message(Form.service)
async def process_service(message: Message, state: FSMContext) -> None:
    await state.update_data(service=get_service_id_by_name("services",
"service_name", message.text))
    service_id = get_service_id_by_name("services", "service_name",
message.text)
    available_dates_data =
get_data(f"available_dates?service_id={service_id}")['available_dates']
    keyboard_buttons = []
    if len(available_dates_data) == 0:
        await message.answer(
            f"Нажаль немає вільних дат для послуги
{html.quote(message.text)}:",
            reply_markup=get_keyboard_start(),
        )
    else:
        for item in available_dates_data:
            available_date = item["available_date"]
            available_time = item["available_time"]
            kb = [KeyboardButton(text=format_date(f"{available_date} o
{available_time}"))]
            keyboard_buttons.append(kb)
        c_bk = [KeyboardButton(text="Відмінити")]
        keyboard_buttons.append(c_bk)
        keyboard = ReplyKeyboardMarkup(
            keyboard=keyboard_buttons,
            resize_keyboard=True
        )
        await state.set_state(Form.available_dates)
        await message.answer(
            f"Зрозумів! Будь ласка, оберіть дати із доступних для
{html.quote(message.text)}:",
            reply_markup=keyboard,
        )
```

Продовження додатка Б

```
@router.message(Form.available_dates)
async def process_available_dates(message: Message, state: FSMContext) ->
None:
    data = await
state.update_data(available_dates=get_id_by_date_and_time("available_dates",
message.text))
    await state.clear()
    await show_summary(message=message, data=data)


async def show_summary(message: Message, data) -> None:
    user_id = get_user_by_tg_id("users", data['user'])["id"]
    name = get_user_by_tg_id("users", data['user'])["name"]
    service_id = data.get("service", "<missing service>")
    service = get_item_by_id("services", "id", service_id)['service_name']
    available_dates_id = data.get("available_dates", "<missing dates>")
    available_date_data = get_item_by_id("available_dates", "id",
available_dates_id)
    available_date = format_date(f"{available_date_data['available_date']} o
{available_date_data['available_time']}")
    url = f"{API_URL}/appointments"
    requests.post(url, json={
        "user_id": user_id,
        "service_id": service_id,
        "date_id": available_dates_id
    })
    text = f"{name}, Ви записані на послугу {service}.\n"
    text += f"- Запис на: {available_date}\n"
    await message.answer(text=text, reply_markup=get_keyboard_start())


@router.message()
async def all_message(message: Message) -> None:
    await message.answer(text="Я Вас не зрозумів, спробуйте обрати із
запропонованих варіантів 👇",
                         reply_markup=get_keyboard_start())


@router.message()
async def check_appointments_and_send_message():
    current_date = datetime.now().strftime("%Y-%m-%d")
    next_day_date = (datetime.now() + timedelta(days=1)).strftime("%Y-%m-%d")
    data = get_data('appointments')
    for appointment in data['appointments']:
        appointment_start = datetime.strptime(appointment['start'], "%Y-%m-
%dT%H:%M:%S").strftime("%Y-%m-%d")
        appointment_start_message = datetime.strptime(appointment['start'],
"%Y-%m-%dT%H:%M:%S").strftime(
            "%d.%m.%Y o %H:%M")
        if appointment['status'] == 0:
            if appointment_start == current_date or appointment_start ==
next_day_date:
                message_text = f"Підтвердіть записаний на
{appointment_start_message}.\n'Послуга {appointment['service']}'"
                await bot.send_message(appointment["telegram_id"],
message_text,

reply_markup=get_keyboard_appointments(appointment['id']))
```

Продовження додатка Б

```
@router.callback_query(AppointmentsCallbackConfirmation.filter())
async def callbacks_num_change(callback: CallbackQuery, callback_data:
AppointmentsCallbackConfirmation):
    appointment = get_item_by_id('appointments', 'id', callback_data.id)
    appointment_start_message = datetime.strptime(appointment['start'], "%Y-
%m-%dT%H:%M:%S").strftime(
        "%d.%m.%Y о %H:%M")
    message_text = ""
    if callback_data.action == "confirm":
        res = requests.post(f"{API_URL}/appointments/{callback_data.id}")
        message_text = f"✅Чекаємо Вас {appointment_start_message} на послуга
{appointment['service']}'"
    if callback_data.action == "cancel":
        res = requests.delete(f"{API_URL}/appointments/{callback_data.id}")
        message_text = f"❌Запис {appointment_start_message} на послуга
{appointment['service']}' Скасовано!"
    await callback.message.edit_text(message_text)


async def main():
    dp = Dispatcher()

    dp.include_router(router)
    # Start the scheduler task
    await asyncio.create_task(check_appointments_and_send_message())

    await dp.start_polling(bot)


if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, stream=sys.stdout)
    asyncio.run(main())
```

Фрагмент коду database.py

```
import sqlite3
import os
from datetime import datetime, timedelta, time

current_directory = os.path.dirname(os.path.realpath(__file__))
DATABASE = os.path.join(current_directory, 'database.db')


def get_db_connection():
    conn = sqlite3.connect(DATABASE)
    conn.row_factory = sqlite3.Row
    return conn


# APPOINTMENTS
def get_all_appointments():
    conn = get_db_connection()
    cursor = conn.execute('SELECT * FROM user_appointments')
    appointments = cursor.fetchall()
    conn.close()
    return appointments
```

Продовження додатка Б

```
def add_appointment_to_db(user_id, service_id, date_id):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('INSERT INTO user_appointments (user_id, service_id,
date_id) VALUES (?, ?, ?)',
                       (user_id, service_id, date_id))
        cursor.execute('UPDATE available_dates SET is_appointments = ? WHERE
id = ?', (True, date_id))
        conn.commit()
        conn.close()

        return True

    except Exception as e:
        print(f"Error adding appointment to database: {str(e)}")
        return False


def update_appointment_in_db(appointment_id, user_id, service_id, date_id):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('SELECT date_id FROM user_appointments WHERE id = ?',
(appointment_id,))
        previous_date_id = cursor.fetchone()['date_id']
        cursor.execute('UPDATE user_appointments SET user_id = ?, service_id
= ?, date_id = ? WHERE id = ?',
                       (user_id, service_id, date_id, appointment_id))
        cursor.execute('UPDATE available_dates SET is_appointments = ? WHERE
id = ?', (False, previous_date_id))
        cursor.execute('UPDATE available_dates SET is_appointments = ? WHERE
id = ?', (True, date_id))
        conn.commit()
        conn.close()
        return True
    except Exception as e:
        print(f"Error updating appointment in the database: {str(e)}")
        return False


def set_status_appointment_in_db(appointment_id):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('UPDATE user_appointments SET status = ? WHERE id =
?', (1, appointment_id))
        conn.commit()
        conn.close()
        return True
    except Exception as e:
        print(f"Error updating appointment in the database: {str(e)}")
        return False


def delete_appointment_from_db(appointment_id):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('SELECT date_id FROM user_appointments WHERE id = ?',
```

Продовження додатка Б

```
(appointment_id,))
        date_id = cursor.fetchone()['date_id']
        cursor.execute('DELETE FROM user_appointments WHERE id = ?',
(appointment_id,))
        cursor.execute('UPDATE available_dates SET is_appointments = ? WHERE
id = ?', (False, date_id))
        conn.commit()
        conn.close()
        return True
    except Exception as e:
        print(f"Error deleting appointment from the database: {str(e)}")
        return False


def get_available_dates_for_service(service_id):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()

        # Fetch duration_hours for the specified service
        cursor.execute('SELECT duration_hours FROM services WHERE id = ?',
(service_id,))
        duration_hours = cursor.fetchone()[0]  # Fetching the value directly
instead of accessing dictionary

        # Fetch available_dates with corresponding available_time
        cursor.execute('SELECT * FROM available_dates')
        available_dates = cursor.fetchall()

        # Convert sqlite3.Row objects to dictionaries and add end_datetime
        formatted_dates = []
        for row in available_dates:
            date = dict(row)
            start_datetime = datetime.strptime(f"{date['available_date']}
{date['available_time']}",
                                               "%Y-%m-%d %H:%M")
            end_datetime = start_datetime + timedelta(hours=duration_hours)
            date['end_datetime'] = end_datetime.strftime("%H:%M")
            formatted_dates.append(date)

        # Filter available slots based on is_appointments
        current_date = datetime.now().strftime("%Y-%m-%d")
        current_time = datetime.now().strftime("%H:%M")
        available_slots = [date for date in formatted_dates if
                           date['is_appointments'] == 0 and
date['available_time'] > current_time or
                           date['available_date'] > current_date]

        # Remove slots overlapping with appointments
        for appointment in formatted_dates:
            if appointment['is_appointments'] == 1:
                overlapping_slots = [slot for slot in available_slots
                                     if slot['available_date'] ==
appointment['available_date'] and
                                     slot['available_time'] <
appointment['end_datetime'] and
                                     appointment['available_time'] <
slot['end_datetime']]
                available_slots = [slot for slot in available_slots if slot
not in overlapping_slots]

        conn.close()
```

Продовження додатка Б

```
        return available_slots

    except Exception as e:
        print(f"Error getting available dates for service: {str(e)}")
        return []


def get_user_by_id(telegram_id):
    conn = get_db_connection()
    cursor = conn.execute('SELECT id FROM users WHERE telegram_id = ?',
(telegram_id,))
    user_id = cursor.fetchone()[0]
    cursor = conn.execute('SELECT * FROM user_appointments WHERE user_id =
?', (user_id,))
    appointments = cursor.fetchall()
    conn.close()
    return appointments


# SERVICES
def get_all_services_from_db():
    conn = get_db_connection()
    cursor = conn.execute('SELECT * FROM services')
    services = cursor.fetchall()
    conn.close()
    return services


def add_service_to_db(service_name, duration_hours):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('INSERT INTO services (service_name, duration_hours)
VALUES (?, ?)',
                       (service_name, duration_hours))
        conn.commit()
        conn.close()
        return True

    except Exception as e:
        print(f"Error adding service to database: {str(e)}")
        return False


def get_service_info_by_id(service_id):
    conn = get_db_connection()
    cursor = conn.execute('SELECT * FROM services WHERE id = ?',
(service_id,))
    service_info = cursor.fetchone()
    conn.close()
    return service_info


def update_service_in_db(service_id, service_name, duration_hours):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('UPDATE services SET service_name = ?, duration_hours
= ? WHERE id = ?',
                       (service_name, duration_hours, service_id))
        conn.commit()
```

Продовження додатка Б

```
        conn.close()
        return True
    except Exception as e:
        print(f"Error updating service in the database: {str(e)}")
        return False


def delete_service_from_db(service_id):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('DELETE FROM services WHERE id = ?', (service_id,))
        conn.commit()
        conn.close()
        return True
    except Exception as e:
        print(f"Error deleting service from the database: {str(e)}")
        return False


# USERS
def get_all_users_from_db():
    conn = get_db_connection()
    cursor = conn.execute('SELECT * FROM users')
    users = cursor.fetchall()
    conn.close()
    return users


def add_user_to_db(tg_id, name, phone):
    print(tg_id, name, phone)
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('INSERT INTO users (telegram_id, name, phone) VALUES
(?, ?, ?)', (tg_id, name, phone))
        conn.commit()
        conn.close()
        return True
    except Exception as e:
        print(f"Error adding user to database: {str(e)}")
        return False


def get_user_info_by_id(user_id):
    conn = get_db_connection()
    cursor = conn.execute('SELECT * FROM users WHERE id = ?', (user_id,))
    user_info = cursor.fetchone()
    conn.close()
    return user_info


def update_user_in_db(user_id, telegram_id, name, phone):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('UPDATE users SET telegram_id = ?, name = ?, phone = ?
WHERE id = ?',
                       (telegram_id, name, phone, user_id))
        conn.commit()
        conn.close()
```

Продовження додатка Б

```
        return True
    except Exception as e:
        print(f"Error updating user in the database: {str(e)}")
        return False


def delete_user_from_db(user_id):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('DELETE FROM users WHERE id = ?', (user_id,))
        conn.commit()
        conn.close()
        return True
    except Exception as e:
        print(f"Error deleting user from the database: {str(e)}")
        return False


# AVAILABLE DATES
def get_all_available_dates_from_db():
    conn = get_db_connection()
    cursor = conn.execute('SELECT * FROM available_dates')
    available_dates = cursor.fetchall()
    conn.close()
    return available_dates


def add_available_dates_to_db(date, time):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('INSERT INTO available_dates (available_date,
available_time) VALUES (?, ?)', (date, time))
        conn.commit()
        conn.close()
        return True

    except Exception as e:
        print(f"Error adding available dates to database: {str(e)}")
        return False


def get_date_info_by_id(date_id):
    conn = get_db_connection()
    cursor = conn.execute('SELECT * FROM available_dates WHERE id = ?',
(date_id,))
    date_info = cursor.fetchone()
    conn.close()
    return date_info


def update_available_date_in_db(date_id, date, time):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('UPDATE available_dates SET available_date = ?,
available_time = ? WHERE id = ?',
                       (date, time, date_id))
        conn.commit()
```

Продовження додатка Б

```
        conn.close()
        return True
    except Exception as e:
        print(f"Error updating available date in the database: {str(e)}")
        return False


def delete_available_date_from_db(date_id):
    try:
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute('DELETE FROM available_dates WHERE id = ?',
(date_id,))
        conn.commit()
        conn.close()
        return True
    except Exception as e:
        print(f"Error deleting available date from the database: {str(e)}")
        return False
```

Фрагмент коду run.py

```
import subprocess
import time

# Start app.py in a separate process
app_process = subprocess.Popen(['python', 'app.py'])

# Wait for a moment to ensure that Flask app has started
time.sleep(2)

# Start bot.py in another process
bot_process = subprocess.Popen(['python', 'bot.py'])

try:
    # Wait for both processes to finish
    app_process.wait()
    bot_process.wait()

except KeyboardInterrupt:
    # Handle KeyboardInterrupt (Ctrl+C) to stop both processes gracefully
    app_process.terminate()
    bot_process.terminate()
    app_process.wait()
    bot_process.wait()
```