МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____Литвиненко О.Є.
"_____"_____ 2021 р.

ДИПЛОМНИЙ ПРОЄКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ЗА НАПРЯМОМ ПІДГОТОВКИ 6.050102 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

Тема: «Телеграм-бот для автоматизації бізнес процесів в ресторанному бізнесі»

Виконавець: студент, група СП-426, Цибульський Микита Олександрович

(студент, група, прізвище, ім'я, по-батькові)

Керівник: к.ф.-м.н., доцент Кучерява Ольга Миколаївна
(науковий ступінь, вчене звання, прізвище, ім'я, по-батькові)

Нормоконтролер: _____ Тупота Є.В.
(підпис)

Київ- 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп’ютеризованих систем управління

Спеціальність 123 «Комп'ютерна інженерія»

(шифр, найменування)

Освітньо-професійна програма «Системне програмування»

Форма навчання денна

ЗАТВЕРДЖУЮ

Завідувач кафедри

О.Є. Литвиненко

«_____» _____ 2021 р.

ЗАВДАННЯ

на виконання дипломного проєкту

Цибульського Микити Олександровича

(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломного проекту «Телеграм-бот для автоматизації бізнес процесів в ресторанному бізнесі» затверджена наказом ректора від « 04 » лютого 2021 р. № 135/ст

2. Термін виконання проєкту: з 17 травня 2021 р. по 20 червня 2021 р.

3. Вихідні дані до проєкту: мови програмування *Java*, *Kotlin*, середовище програмування *Inteliji IDEA 2021.1.1*, програмний веб-інтерфейс *Telegram Bot API* .

4. Зміст пояснювальної записки:

1) Тут писати назви розділів ДП

2)

3)

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) Тут навзи граф матеріалів

2) _____

3) _____

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1.	Аналіз джерел за темою дослідження. Розроблення плану дипломного проєкту	17.05.2021-19.05.2021	
2.	Затвердження плану дипломного проєкту та проведення консультацій з науковим керівником, щодо наповнення пояснювальної записки	20.05.2021-21.05.2021	
3.	Дослідження засобів для роботи з телеграм-ботами	22.05.2021-24.05.2021	
4.	Розробка архітектури чат-бота для ресторанного бізнесу	25.05.2021-26.05.2021	
5.	Створення чат-бота для ресторанного бізнесу	01.06.2021-10.06.2021	
6.	Написання пояснювальної записки	11.06.2021-15.06.2021	
7.	Підготовка демонстраційного матеріалу	16.06.2021 20.06.2021	

7. Дата видачі завдання: «_17_» _травня_ 2021 р.

Керівник дипломного проекту _____ к.ф.-м.н., доц. Кучерява О. М.

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Цибульський М. О.

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту «Телеграм-бот для автоматизації бізнес процесів в ресторанному бізнесі»: 54 с., _ рис., 1 табл., 0 літературних джерел, 1 додаток.

чат-бот, *telegram bot, java, kotlin, spring boot, back-end application*

Об'єкт дипломного проєктування – чат-бот для меседжеру *Telegram*

Предмет дипломного проєктування – створення чат-бота для для автоматизації бізнес процесів.

Методи проєктування – розробка чат-бота для автоматизації бізнес процесів в ресторанному бізнесі.

Методи дослідження – технології стоворення чат-ботів, методи об'єктно-орієнтованого програмування, технології управління базами даних *Spring Data, MySQL, Hibernate*, порівняльний аналіз ринку чат-ботів.

Здійснено огляд теорії створення чат-ботів на основі готових бізнес процесів та існуючих чат-ботів, що виконують обробку та аналіз замовлень користувачів, здійснено порівняльний аналіз їх функціональних можливостей; ознайомлено з принципипами побудови додатків для архітектури серверного веб-додатку; реалізовано програму для обробки запитів чат-бота по формуванню та обробці процесів ресторного бізнесу.

Матеріали дипломного проекту **рекомендується використовувати** при самостійному вивченні теми будування чат-ботів, у навчальному процесі фахівців з створення серверно-орієнтованих програмних застосунків та при вивченні об'єктно-орієнтованого програмування, а також у проєктах пов'язаних автоматизацією бізнес процесів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

CI/CD – Continuous Integration / Continuous Deployment

API – Application Programming Interface

MR – Merge Request

ETH – Ethereum

MVC – Model View Controller

UX – User Experience

MD5 - Message Digest 5

ВСТУП

Актуальність теми. Ресторанний бізнес в умовах карантину опинився в умовах в яких необхідно пристосовуватись до навколишніх обставин швидше за конкурентів, щоб залишитись в ніші бізнесу. Перш за все, стресом для ресторанних бізнес-систем стала доставка клієнтам товарів. Для великих міст проблема доставки вирішується вже протягом останіх декількох років в вигляді кур'єрських сервісів, тоді яку більшості населених пунктів України підтримки сервісів і попиту на послуги доставки до початку карантину не було, через те, що проблема зручності доставки товарів стала актуальною саме під час закриття міст на закриття, коли ресторани працюєть лише у режимі доставки. В більшості ресторанів невеликих міст досі використовується гаряча лінія або ж сайт, через який людина може зробити замовлення. Проте у більшості випадків для замовлення необхідно звертатись за телефоном до оператора, який буде обробляти замовлення та рекомендувати товари з меню у випадку, якщо для клієнта не принциповим вибір. Такий метод несе велику кількість витрат в порівняні з автоматичним виконанням даних запитів, через те що в процесі спілкування з клієнтом може здійснені помилки в обробці замовлення та клієнт не отримає необхідної інформації про знижки на вибраний перелік товарів, не вказано потрібний клієнту товар, не буде наявним вид оплати, який задовільнив би клієнта, бізнес буде витрачати кошти для того, щоб повернути клієнта чи поновити втратити репутацію закладу через некомпетентність оператора а також необхідно не забувати, що цей процес відіймає час у людини, яка буде в ручному режимі обробляти замовлення.

Основою для створення телеграм бота для автоматизації бізнес процесів пов'язаних з доставкою товарів та обробкою замовлень є серівіси доставки, розроблені компаніями які стали одними найшвидш зростаючими протягом періоду карантину у зв'язку з *Covid-19.*

Найбільшою перевагою в порівнянні з сервісами, які надають послуги доставки через меседжери є нестандартний підхід в побудуванні бота, що є більш схожим на звичайні мобільні додатки, та формує миттевий доступ до меню та замовлення. Якщо ж порівнювати з наведеним в першому абзаці процесом обробки замовлень,

за допомогою дзвінків, такий спосіб принесе в рази більший прибуток, звадяки таким можливістям: переглянути акційні пропозиції, можливість перевірити актуальність ціни товару, його наявність описи товарів, для кращого уявлення покупки та приваблення нових клієнтів.

Даний проект є преспективним так дозволяє майже миттево замовити товар без складних процесів реєстрації та завантаження додатків, які не завжди є потрібними для користувача, що не часто робить замовлення в сервісі.

В східних країнах вже давно оцінили зручність даного способу поширення інформації та використовують його як заміну сайтам та мобільним додаткам в звичайному їх розумінні.

Мета і завдання дипломного проєкту.

Метою даного проєкту є проектування та розробка комп'ютерної програми для проведення процесу обробки замовлень для доставки.

Об'єкт дипломного проєктування – чат-бот для меседжеру Telegram

Предмет дипломного проєктування – створення чат-бота для для автоматизації бізнес процесів.

Методи проєктування. Технології ствренння чат-ботів було використано для стоврення та побудування систем взаємодії з клієнтом; порівняльний аналіз застосовується у першому та третьому розділах для виділення особливостей або переваг того чи іншого програмного продукту; методи об'єктно-орієнтованого програмування – використано при розробці функціоналу чат боту та формуванні архітектури програмного застосунку. Технології управління базаии даних було використано для сторення редагування структури та взаємодії з структурами даних збережених в реліаційних базах даних.

Здійснено огляд теорії електронної обробки даних та існуючих чат ботів; здійснено порівняльний аналіз їх функціональних можливостей; розроблену структуру моделі баз даних проєкту; розглянуто можливості взаємодії через *API* за допомогою *Java* в парі з *Kotlin* кодом; досліджено особливості побудування систем таблиць баз даних; досліджено засоби і методи роботи з асинхронними викликами до даних; досліджено фреймворки для побудування серверно-орієнтованих

додатків; розроблено структуру програми для роботи над бізнес процесами всередині чат-бота; реалізовано чат-бота для авторматизації бізнес процесів.

Практичне значення отриманих результатів. Результуючі методи можуть використовуватися для самостійного вивчення теми будування чат-ботів, у навчальному процесі фахівців з створення серверно-орієнтованих програмних застосунків та при вивченні об'єктно орієнтовного програмування, а також у проєктах пов'язаних автоматизацією бізнес процесів. В результаті дипломного проєктування отримано чат-бот для автоматизації бізнес-процесів. Враховуючи обмежену кількість аналогів системи та переваги використання чат-ботів над веб застоунками, чат-бот такого плану є унікальним для масового вмикористання та може бути пристосований до будь якого бізнесу, де необхідно отримувати товари з доставкою. Функціонал, описаний у програмних модулях проєкту може бути модифікований або розширений без значних зусиль.

Особистий внесок випускника. Всі результати, представлені у дипломній роботі, отримані випускником особисто. Були прийняті рішення щодо архітектурних особливостей та побудовано власний концепт роботи в чат-ботах базовний на досвіді розробки мобільних додатків.

Апробація отриманих результатів. Практичні аспекти роботи з чат-ботом пройшли апробацію при тестуванні на реальному бізнесі з обмеженою кількістю осіб.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ЧАТ-БОТІВ ТА ЇХ СТРУКТУРНИХ ОСОБЛИВОСТЕЙ

1.1 Історія чат-ботів

Чат бот – це комп'ютерна програма, за допомогою якої можна здійснювати комунікацію, на основі заданого алгоритму в чатах. Дана системи є хорошим прикладном «людино – компютерних» систем. Вперше даний термін було вжито в 1994 році Майклом Маулдіном, творцем першого чатерботу (з анлг. *ChatterBot*) *Julia*, щоб описати розмовні програми, які дозволяють комунікувати між ботом та людиною в текстовому вигляді.

На сьогодні, це поняття дещо видозмінилось адже прогрес не стоїть на місці та вже існують програмні засоби щоб зробити на основі чатів додатки з зручним для користувачів інтерфейсом на основі кнопок, команд, голосового вводу, щоб видати клієнту необхідний йому матеріал чи інформацію в будь- якому з доступних видів зображення інформації.

Боти в чатах вже давно стали невідємною частиною будь якого меседжера. На сьогоднішній день, найпопулярнішими соціальними-мережами, які дозволяють створювати на їх основі чат-ботів є *Telegram, Viber, Facebook, Discord, VK, WhatsApp, Instagram* та меседжери, число яких з кожним днем зібльшується, проте дехто з них вже давно залишив позаду рушту, і зараз мова йде про китайську соціальну мережу *WeChat*, де на теперішній момент, меседжер пішов набагато далі та дав можливість розпосюджувати та продавати товари через аналог теперішніх чат-ботів, міні додатки. *Telegram* в свою чергу в весінньому релізі 2021 року, добавий перший крок в цьому напрямку, представивши розробникам можливість здійснювати покупки всередині меседжера, за допомогою зручного інтерфейсу.

Розробка ботів буває не лише у чатах, а останім часом широко застосовується для взаємодії з клієнтами на сайтах, в вигляді чату підтримки 24/7. Для бізнесу це можливість забезпечити безперерву підтримку клієнтів, на основі популярних питань чи зменшити взаємодію клієнтів з персоналом при високому навантаженні на системи.

1.2 Типи чат-ботів

Чат-ботів поділяють за такими критеріями:

- за форматом взаємодії
- за призначенням
- за складністю алгоритму

1.2.1 Чат-боти за форматом взаємодії

Чат боти в початковому вигляді мають такі формати інтерфейсу взаємодії з користувачем:

- текстовий інтерфейс
- кнопковий інтерфейс
- аудіо-інтерфейс

За допомогою текстового інтерфейсу можна створити бота базованого на взаємодії за допомогою команд, що для користувача виглядає як відправка заготовлених фраз чи слів, які мають спільний символ як префікс, для того щоб правильно відображатись та підказувати користувачу, що це команда. Для телеграм ботів цим символом є “/”.

Кнопковий інтерфейс, підтримується новими меседжерами для забезпечення роботи контекстних меню, та може містити собі логіку переходів за посиланнями, чи змінювати стан бота в залежності від складності алгоритму. Більшість чат-ботів використовує його, як додатковий елемент вазаємодії з користувачем поміж роботи з командами користувача та його вводу. Для людини, яка розробляє даних ботів різниця між командами та обробкою натискань на кнопки не є суттєвою, так як за занавісою фронтенду, все так само оброблються запити користувача, як текстові команди.

Наступним способом комунікації серед чат-ботів є аудіо взаємодія з контентом. Для того щоб розробити такі системи, потрібно враховувати помилки, такі як, неправильна обробка чи розпізнавання аудіо запиту від користувача. Порівняно з двома попередніми, де сценарії запитів є фіксованими, та можуть бути протестовані повністю, сценаріїв в випадку аудіо-інтерфейсів може бути бізліч тому й основна робота полягає в тому, щоб навчити штучний інтелект, передбачати те що має на потребує користувач, переводити в зрозумілий для машини алгоритм,

та видати релевантну інформацію. Більшість чат-ботів в загальному використовує змішаний інтерфейс взаємодії, так як таким чином досягається найкращий досвід користування, через його схожість з сучасними мобільними додатками. При достатньому досвіді в розробці, можна створити «клієнт-серверний» додаток, який буде створювати ілюзії роботи зі звичайним андроїд-додатком проте з привязкою до можливостей телеграму для відображення контенту чи іншого меседжеру на якому розробляється чат-бот.

1.2.2 Типи чат-ботів за призначенням

Ботів поділяють за призначенням так:

- боти комунікації
    - боти-асистент
    - боти підтримки клієнтів
- боти надання послуг (функціональний)
    - транзакційні боти
    - лід-бот
- інформаційні боти

Чат-бот комунікації – це чат-бот, базована на концепції спілкування між клієнтом та системою, спрямована на підтримку бізнес системи, відповіді на питання клієнта, надавання зворотнього з'вязку, з'єднання з експертами, та інші взаємодії пов'язані з комунікацією та наданням консультацій клієнтам. Прикладом чат боту комунікації є чати фінансових-установ в *Telegram*: *Monobank* (*@monobankbot*), *PrivatBank* (*@PrivatBankBot*), *Kuna* (*@KunaSupport_bot*). До даного підтипу ботів можна віднести будь якого бота госового помічника, таких як *Google Assistant*, *Amazon Alexa*, *Yandex Alissa* та інші. Принцип роботи даних ботів базується на живому спілкуванні та використовує для роботи як і алгоримічні структури (послідовні відповіді на питання) так і методи пошуку на основі штучного інтелекту, для видачі найбільш релевантної інформації згідно з запитом користувача.

Чат-бот надання послуг – це чат-бот, що виконує певний бізнес функціонал повязаний безпосередньо з наданням послуг та супроводженням клієнтів протягом цього процесу.

Транзикційний чат-бот - це чат-бот, який виконує різноманітні транзакції, розміщення замовлень, резервування, переказ грошей і так далі. Чат боти такого типу найбільш поширені серед користувачів, так як надають послуги в найбільш зручному для користувачів вигляді, та дозволяють без заєвих зусиль отримати необхідні послуги, це може бути купівля товарів, отримання промо-кодів, акційних пропозицій, обміну валют та й взагалі може формувати самостійний додаток, всесередені іншого меседжера. Прикладами можуть слугувати *@kunacodebot* - бот для обміну кріптовалюти між користувачами, *@AnyCashBot* –гаманець валют криптовалют, базований на Binance біржі. *@railwaybot* – чат-бот для відслідковування та купівлі квитків на потяги, який також містить в собі ознаки наступного підтипу чат-ботів.

Інформаційний чат-бот – це чат-бот, для надання інформації за запитом користувача. Даний тип ботів є підтипом функціональних ботів, проте відрізняється тим, що надає інформацію за певними запитами, та не надає послуг як таких, окрім як підтримки існуючих систем. А саме може представляти користувачу інформацію про обробку замовлення, статус системи, сповіщати клієнта про неполадки в сервісіх і так далі. Прикладом такої системи може бути *@ethermineinfobot* – бот, для отримання інформації про роботу робочої одиниці на пулі задач з майнінгу ефіру (*ETH*) з сайту *ethermine.pool*.

@GmailBot – бот, для перенаправлення пошти з звичних поштових скриньок в бота, який в той же час вміє фільтрувати спам, запам’ятовувати необхідні фільтри, та відповідати на листи.

1.2.3 Чат-боти за складністю алгоритму

Чат боти, як було згадано вище можуть бути базовані, на основі стійких систем, для яких буде прописаний повний функціонал та всі відгалуження програми можуть бути протестовані, на етапі розробки, так і боти які матимуть необмежений перелік команд, з якими буде взаємодіяти користувач, а саме аудіо-повідомлення, звідки буде отрмано загальну інформацію на основі якої будуть робитись припущення про відповідність результату видачі до запиту користувача. Такі

системи складаються в результаті розробки великих корпорацій, наведених в прикладах на початку опису. Відносно простим в розробці можна назвати лінійний алгорим взаємодії з користувачем, проте, як і з будь яким продуктом, все впирається в гнучкість архітектури та покриття тестами таких додатків.

1.3 Особливості розробки чат-ботів для *Telegram*

Для розробки бота використовується веб-інтерфейс розробника, який може представляти для цього бібліотеки, що будуть обробляти інформацію, яка надходить з *API* для взаємодії між користувачем та додатком. До розробки таких бібліотек може долучить будь який бажаючий вдосконалити спосіб взаємодії з кінцевими-точками (з англ. *end-points*) меседжера.

Для меседжера Telegram, існують бібліотеки написані на більшості популярних мов програмування, включаючи *C++, Java, Ruby, Kotlin, JavaScript, PHP, Node Js, Rust, Python, Swift, Go* та інші мови програмування. При цьому вони активно розвиваються, і протягом останього року було добавлено більше п’яти мов програмування і з десяток бібліотек від стороніх розробників. В більшості випадків бібліотеки не дають додаткових можливостей окрім тих, які визначені розробниками *Telegram*, проте є і ті бібліотеки, які значно розширяють їх функціонал, обходячи обмеження накладені згідно політики безпеки *Telegram*, а сама можливість отримати всіх учасників груп чи каналів, перевірити підписки на канали і так далі, такі можливості дає на теперішній момент лише одна з відомих мені бібліотек для *Python – Telethon*.

1.3.1 Отримання токена бота

Для отримання інформації чи поширення нею через *API* розробника, необхідно для початку скористатись токеном розробника, який можна отримати для кожного з ботів у офіційного бота *Telegram BotFather* через процес створення нового бота, який детально описано на офіційні сторінці розробника, та оновити за необхідності, якщо, наприклад, ви припускаєте, що ці дані могли бути поширені випадково на репозиторії, де ви зберігаєте свій код. Бажано не допускати таких ситуацій, так як це дасть змогу злоумисникам використати аудиторію вашого боту, для того, щоб в кращому варіанті зібрати мінімальну інформацію про ваших користувачів і в найгіршому - зіпсувати репутацію бренду та відібрати у клієнтів

бота гроші, через різноманітні способи, такі як проведення оплати черз криптовалютні гаманці, в яких відслідковування власників таких операцій не є можливим через внутрішню структуру всієї системи. Тому обов'язково необхідно додавати файли, де зберігається токен бота в *.gitignore* файл.

1.3.2 Процес отримання даних про події

Для того щоб отримати дані від телеграм боту, існує два загальноприйняті, та досить детально описані процеси, які використовуються не лише в отриманні даних від ботів, а й у таких завданнях, як оновлення статусу *GitLab MR* у зв'язці з *CI/CD Jenkins* та інші. Першим є простий і в той час ефективний спосіб є довге опитування (*long pooling*) серверу, яке полягає, у відправлені запитів завчасно визначену кінцеву-точку *Telegram* з заданою періодичністю. Таким чином розробник

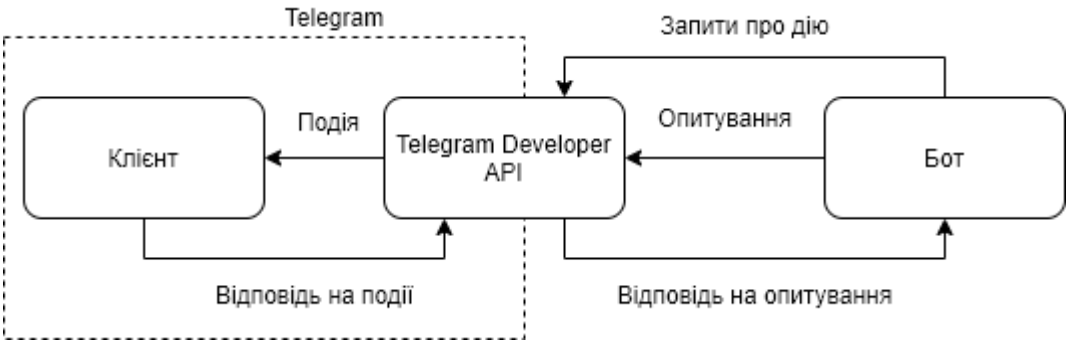

Рисунок 1.1 – Схема опитування серверу

звільняється від відповідальності за шифрування контенту, який передається з серверів, так як за замовчуванням весь трафік передається через *https* шлюз. На противагу плюсів, в даного способу є ряд мінусів, які накладає обмеження на роботу такого способу: переший і найочевидніший мінус - це нерівномірне розподілення ресурсів згідно з навантаженням на бота. Для бота, який буде працювати за таким способом, навантаження на мережу постійно буде сталим, адже щоб отримати актуальну інформацію, про запити від користувача, необхідно постійно відправляти запити, через це можливий варіант, коли не буде достатньо ресурсів, на те щоб обробити запит, який прийде з запізненням, через те, що утвориться черга запитів в нашому пулі.

Найкращим з точки зору оптимальності розподілення ресурсів є спосіб отримання інформації на кінцеву точку нашому сервера – *webhook* (веб-перехоплювач). Спосіб, полягає в отримані інформації на кінцеву точку в вигляді

публічного адресу, та відправка інформації як зворотні події на *end-point* вказаний для *Telegram* бота. Даний спосіб є абсолютно протилежним попередньому варіанту, так як в цьому випадку ми виконуємо роль приймача, та лише очікуємо на оновлення від клієнта, й в разі отримання події відповідаємо на його запити згідно заданого алгоритму, що дає можливість використовувати меньше ресурсів та економити на хмарних технологіях для розміщення наших серверів. Мінусом є порівняно складніший спосіб налаштування бота, для отримання все тієї ж інформації, що і в попередньому випадку. Далі буде описано цей процес.

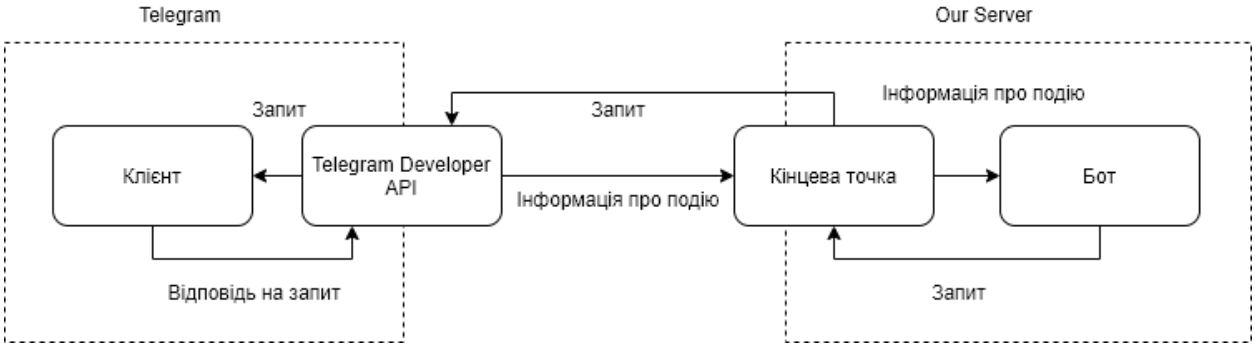

Рисунок 1.2 – Схема роботи *webhook*

1.3.2.1 Створення бота на основі *Long Pooling* методу отримання даних

Процес створення ботів на основі опитування, є простим з точки зору архітектури та використовуваних технологій, так як, для цього необхідно відправляти з заданою періодичністю запити на сервер *Telegram*. Цей спосіб може бути реалізований за допомогою базових можливостей *TelegramBots* бібліотеки для мови програмування *Java*. Для розробника для створення бота необхідно створити об’єкт класу опитування та записати ініціалізаційні поля для синхронізації з серверною стороною *Telegram*.

class OurTelegramLongPoolingBot extends TelegramLongPollingBot

Для роботи з таким ботом обов’язковим буде приписання таких методів як:

getBotUsername – вертає псевдонім бота

getBotToken –повертає токен бота, який було отримано в BotFather

onUpdateReceived – метод, який отримує події та в якому відюується взаємодія з користувачем

Для запуску бота потрібно за допомогою методу екземпляру класу *TelegramBotsApi registerBot* зареєструвати на бота *Telegram API* передавши йому сконфігурований клас який буде наслідуватись від класу *TelegramLongPoolingBot*

```
ApiContextInitializer.init();
TelegramBotsApi telegramBotsApi = new TelegramBotsApi();
telegramBotsApi.registerBot(new OurTelegramLongPoolingBot());
```

Об'єкт *Update* містить в собі інформацію про запити, які надсилає користувач. Наступним кроком після отримання об'єкту ми маємо визначити який саме контент містить цей об'єкт. Для цього використовуються методи цього класу, які перевіряють наявність даних певного типу.

hasMessage – перевіряє наявність класу повідомлення *Message*

hasCallbackQuery– перевіряє наявність класу *CallBackQuery*, який містить інформацію про зворотній виклик кнопок, які можна використовуати для побудування інтерфейсу користувача.

hasEditedMessage – перевіряє наявність класу *EditedMessage*, який приходить у ви падку, якщо повідомлення було відредаговано користувачем, чи іншим ботом.

А також інші методи для класів даних побудовані за схожим принципом. Для відправки повідомлень використовується метод *execute* в парі з об'єктом методу *Telegram Bot*-а, що містить в собі всі данні методу, такі як текст, *id*-чату, тип розмітки (*Markdown, HTML*) та інші в залежності від типу інформації об'єкта.

1.3.2.2 Процес створення бота на основі методу отримання даних *Webhook*

Для створення боту даним способом необхідно отримати кінцеву точку, та зареєструвати її для отримання даних на неї від *Telegram Bot Api*. Після чого неохідно зчитувати дані в форматі *json* та перетворені його в об'єкти, з заданими документацією полями, що дадуть нам можливість взаємодіяти з ботом.

Процес створення кінцевої точки, полягає в використанні утиліти з назвою *ngrok*, що дозволяє створити в безкоштовному варіанті програми не більше одного публічного адресу, який буде посилатись на порт серверу з якого працює утиліта.

При цьому адрес є статичним лише в платній версії, що значить для нас як

розробників, те що, при перезапуску серверу, необхідно встановлючати новий адрес, який генерується кожного разу інший, та може бути зчитаний з конфігураційного файлу.

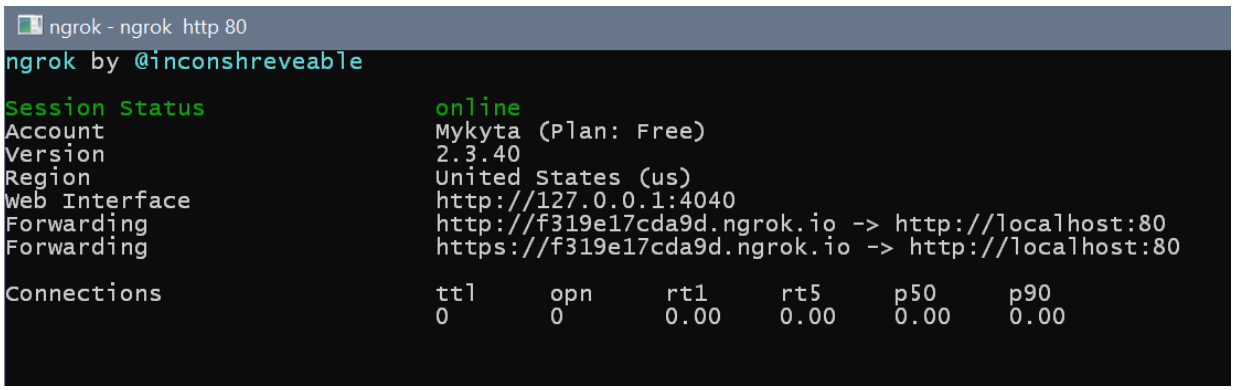

Рисунок 1.3 – Скріншот роботи утиліти *ngrok*

Для роботи з відносно невеликими проектами, цього більш, а ніж достатньо для швидкого підняття програмного забезпечення на серверах хмарних технологій, типу *Google Cloud, Amazon AWS, Microsoft Azure* та інші. Після створення адресу в утиліті, її можна перевести в режим фонового процесу з логуванням помилок в файл, що буде зберігатись на загальному накопчувачі, та не буде стосуватись нашого контейнеру напряму. При створенні адресу буде вказано порт, через буде виконуватись взаємодія з сервером *Telegram* і на який буде подаватись інформація про зміни у клієнті бота. Даний порт, ми вказуємо в конфігурації додатку в якому працюватиме бот на основі *Spring Boot*, та на який буде приходити інформація про зміни стану бота.

Більшість роботи з парсування та валідації даних з обробкою помилок, бере на себе бібліотека, тому для розробника залишається робота з заготовленим об’єктом, який містить в собі інформацію про подію, яку відправляє *Telegram Bot API*.

Для роботи з даними об’єктами на мові програмування *Java*, можна використати *Spring Boot* фереймворк, який дозволяє зментиши навантаження на разробника, за допомогою конфігурації робочого простору для роботи напряму з даними які приходять на порт, з якого будуть зчитуватись дані, таким чином, маючи в розпоряджені адрес, який посилається на *Telegram Bot API*, та виконує функцію proxy для даних на додаток, який працює за заданим портом. Дані можуть бути отримані в додатку, та оброблені згідно з заданого алгоритму.

Як і в будь якому *back-end* додатку побудованого на *Spring*, для отримання даних необхідно створити *Controller* помічений анотацією *@RestController*, на який приходять *POST* запити з даними та який буде повертати callback для серверу, як відповідь на даний запит в вигляді об'єкту який наслідується від *BotApiMethod*, що містить інформацію про дію, яку необхідно виконати клієнту. Далі ми оброблюємо дані, як і в випадку з методом отримання даних - довгого опитування.

1.4 Зберігання чутливої іноформації

Для зберігання чутливої інформації необхідно створити умови, коли доступ до неї матиме лише людина, що розробляє бота, та не надавти можливості, для отримання ззовні. Для більшості випадків, достатьо буде окремого файлу ресурсів, який містить цю інформацію. Зчитування цього файлу та підстановка значень токену, алреси, порту додатку відбувається на етапі конфігурації, та не може бути змінене в процесі роботи додатку. Для налаштування використовується клас конфігурації помічений анотацією *@Configuration*, що вказує на те, що його необхідно виконати перед створенням всіх *Beans* (об'єкти *Spring* додатків, які можуть бути передані в будь яку точку нашого бота за допомогою *@Autowired*, що посилатиметься, за замовучанням, на об'єкт створений за патерном *Singleton*, а отже і не буде створюватись в новому екземплярі при повторному посиланні на *Been*), нашого додатку. В тому ж класі і виконується підстановка та формування головного об'єкту в вигляді *Bean*, що буде реалізовувати обробку класу *Update*, та через який можна буде виконувати методи *Telegram* через метод *execute*.В випадку, якщо є необхідність зберігати дані, недоторканість яких є пріорітетною для запуску всередині додатку, є можливість перевірки за хешами файлів-ресурсів за допомогою *MD5* функції, який гарантуватиме незмінність токенів. Таким чином можна зберігати токени на сервері, який розміщений в захищеному місці, та за допомогою простого мікросервісу діставати дані на сервері, щоб перевіряти незмінність файлів за таблицею оригінальних хешів для файлів реурсів.

Складнішим та більш затратнішим варіантом є спосіб в якому дані будуть передаватись в зашифрованому вигляді і отримання цих даних зловмисником не будуть мати значення, без отримання ключа для даного способу шифрування. Тому для доступу до реальних даних, необхідно отримати доступ до обох серверів, що

вже є непідсильною задачею, враховуючи сучасні методи захисту хмарних технологій. І як останій пункт в забезпеченні безпеки ресурсів, можна налаштувати перешифрування даних за часом, та привязати зміну до часу, як це реалізовано в двохфакторній авторизації, де перевірка хеш сум змінюється кожний заданий проміжок часу. Підсумуймо, для забезпечення найкращої захищеності даних, необхідно використати варіант, в якому використовується розділення відповідальності за збереження інформації на мікросервіси, що розташовані не на одному сервері, проходить шифрування даних з перешифруванням з заданим проміжком часу, передача даних відбувається лише за протоколом *https* і як результуючий варіант проводиться перевірка хеш сум, для кожного з файлів згідно заготовленої *hash*-таблиці.

1.5 Принцип побудови бота на основі станів

Проектуючи архітектуру бота, найпростішим є варіант послідовної відповіді на запити користувача, та робота без залежностей до стану користувача. Такий спосіб є прямолінійним, та в багатьох випадках є доцільний, так як не вимагає бази даних, для зберігання інформації про користувача. Що зменьшує затрати на розробку та відладку, проте несе ряд недоліків:

- користувач не отримує послідовності в діях і більшість дій будуть побудовані на одній дії зі сторони бота
- розробник не має можливості відслідковувати роботу користувача, та представляти можливість переходу між станами в межах одного користувацького-інтерфейсу (як приклад, інтерфейсом взаємодії може бути меню вбудованої клавіатури в повідомленні)

Для уникнення ситуацій, в яких нема можливості побудувати дружнього до користувача інтерфейсу, вводеться поняття станів користувача.

Стан користувача – позначення положення користувача в рамках додатку.

Використання станів, дає можливість будувати залеженості відносно один одного та пов’язувати дії користувача з певними станами. Так наприклад, для вводу даних користувачеві при переході в стан «Запит на введення» буде запропоновано ввести дані, які будуть привязані до конкретної поведінки, та оброблені згідно з

цією поведінкою. Це може бути заповнення профілю користувача в вигляді відповідей на питання бота з можливістю вводу через кнопки (запропоновані варіанти для даного типу вводу) чи як ввід інформації будь якого доступного виду (текст, аудіо, відео, локація і так далі).

Таблиця станів користувача зазвичай зберігається, в базах даних, та містить число стану, відповідно до перерахування (*enum*) наведеного в додатку. Кожному зі станів, відповідає один або більше класів, які вказують послідовність переходів між ними, та містять дані про попередній та теперішній стан, для реалізації «чат-боту одного повідомлення», яке буде описано далі.

Згідно з класифікації чат-ботів, боти можуть бути побудовані на текстових командах, кнопковому інтерфейсі та іншими. Проте на практиці зазвичай використовується перші два, де кожному із способів взаємодії відведена важлива роль при побудуванні зручного та зрозумілого користувачу інтерфейсу. Перший спосіб, несе функцію моментальної зміни стану користувача для досягнення бажаного результату, в незалежності від того на якому етапі в структурі чат-бота находиться користувач, і наступний, кнопковий інтерфейс, використовується для послідовного переходу між станами завчасно визначеного розробником. З досвіду користування було зроблено висновок, що використання вбудованих клавіатур є найзручнішим та найзрозумілішим способом взаємодії, так як беручи аналогію з звичними додатками для мобільних платформ, користувач, зазвичай вивчає інтерфейс, та відповідно очікує, що якщо він пройде певні кроки в додатку, він досягне бажаного результату. Таким чином, використання команд для навігації лише руйнує концепцію послідовної навігації зважаючи на те, що користувач переміщуючись за допомогою команд, втрачатиме омновне меню, яке в звичному поданні буде змінюватись відносно взаємодії з користувачем. Таким чином було виведено, іще один змішаної взаємодії двох способів, який містить інформацію в одному повідомленні, та змінюється в залежності від команд, станів, вводу користувача. Принцип роботи полягає в збереженні першого повідомлення від бота та зміна лише його наповнення з контентом, який відповідає стану користувача. При цьому залишається можливість для вводу в моменти коли це необхідно для зчитування інформації, чи отримання файлів від користувача. Всі повідомлення в

такому способі видаляються одразу після отримання серверною стороною, так як такі повідомлення мають сенс лише для конкретного стану, та їх використання, лише відволікає користувача від головного контенту, та не дає можливості отримувати той *UX*, як задумано розробниками чат боту.

Таким чином, даний концепт було названо - «чат-бот одного повідомлення». Згідно з яким, ми маємо обмежуємо зберігання інформації в чаті, та обов'язково маємо давати можливість видаляти повідомлення, без руйнування користувацького досвіду базованому на роботі звичайного додатку.

1.6. Висновки до розділу

В розділі було опрацьовано та виділено головні особливості систем роботи з чат-ботами та наведено приклади для кожного з типів, що було описано. Було розглянуто процес створення від отримання токена, до розподілення стани користувацьокого інтерфейсу з застосуванням методу розробки бота, який має назву «чат-бот одного повідомлення». Було описано способи взаємодії з чат-ботом з використанням технологій, такий як *Webhook* та *Long Polling.*

РОЗДІЛ 2. ПОБУДУВАННЯ ЧАТ-БОТУ

2.1. Архітектура чат-бота ресторану

Архітектура бота, повністю базується на концепті «чат-бота одного повідомлення», що накладає деякі обмеження, так як необхідно орієнтуватись на стани користувача і в той же час дозволити управління за допомогою команд. Іншим ключовим моментом, для будування архітектури стало використання класичного *Spring Boot* додатку як основи і за замовчуванням містить особливості повязані з архітектурою *MVC,* в якій є такі компоненти як модель-даних, через який змінюється стан нашого клієнта на серверній стороні, контролер, який виконує направляючу функцію по алгоритму, та представлення – те що ми віддаємо користувачу, як відповідь, у вигляді певної події.

Такими об’єктом в додатку є *Controller – RestaurantBot*, що виконує функцію приймача та перенаправлювача інформації по додатку в зв’язці з головним оброблювачем інформації *TelegramFaçade*, *Model* – обробники станів (*handlers*), які наслідуються від головного класу *InputHandler*, та на який потрапляємо для зміни стану бота та представлення інформації для користувача через інтерфейс чат-боту *Telegram,* після чого буде здійснюватись наступний запит на контроллер. Дана схема є дещо спрощена, адже повина відображати головну суть архітектурного шаблону. Для того, щоб краще зрозуміти весь процес обробки, даних та зміни стану, далі буде описаний загальний алгоритм роботи обробки даних які потрапили на *Controller*.

Рисунок 2.1 – Схема аріхітектурного шаблону *MVC*

Після відправлення команди на кінцеву точку бота, отримується інформація про повідомлення, яке перенаправляється на *TelegramFacade*, в якому через метод, що розподіляє згідно з внутрішнім контентом об’єкт *Update* на метод *handleInput* з

параметром класу *Message*, що відповідає нашій команді. Далі проходить перевірка на відносність до можливих команд та згідно цього проініціалізовується першочерговий стан користувача в боті, якщо про це свідчить команда. Зазначення початкових даних, полягає в зчитуванні інформації доступної з об'єкта який прийшов як параметр об'єкту *Update*, та записання цих даних в базу даних (робота з якою буде описана в підрозділі «Робота з базою даних»), через сервіс роботи з таблицями даних користувачів. Далі проходить зчитування стану користувача з бази даних та перенаправляється на метод, який вирішує в який саме обробник подій має попасти дане повідомлення. Після чого виконується алгоритм який міститься в методі handle класу екземпляру обробника та повертає метод бота, який необхідно кваліфікувати, як «Представлення» для клієнта

Основною архітектруною одиницею для формування станій являється класи помічені анотацією *Spring - @Component*, а також наслідуються від абстрактного класу- *InputHandler*. Дані класи відносяться до пакету *handlers* та виконують обробку кожної з подій, що надходить до серверної сторони. Компоненти системи формуються в список, з якого ми можемо витягнути назву стану за допомогою методу *getHandlerName*, та співзіставивши його з тим, що маємо за стан бота, привязаного до користувача, ми можемо коректно відреагувати на запит через метод *handle*, з параметром, який несе інформацію про подію передану корисувачем.

Абстрактний клас містять в собі такі методи:

Отримання поточного стану користувача, як одне з найменувань в *BotState*

public abstract BotState getHandlerName();

Отримання стану користувча, в який він може попасти за допомогою кнопки *back*, яка присутня у всіх станах користувачів, окрім початкового, та головного меню, що зумовлене логікою переходів всередині додатку.

public abstract BotState getPreviousHandlerName();

Методи обробки подій, змінення стану та повернення відповіді для користувача.

public abstract BotApiMethod handle(Message message);

public abstract BotApiMethod handle(CallbackQuery callbackQuery);

Метод для формування відповіді на запит клієна, в вигляді об'єкту, що містить

інформацію про необхідний метод чат-боту - *BotApiMethod*, в більшості випадків використовується спосіб редагування повідомлення – EditMessageText, для першого повідомлення всередині бота використовується відправка нового повідомлення, адже редагувати необхідно реальне повідомлення, і створення нового при запиті до неіснуючтого не відбувається

public BotApiMethod getReplyMessage(Message inputMessage, String textTag, List<List<InlineKeyboardButton>> customKeyboard, boolean createMessage, String errorTag)

Метод для отримання клавіатури користувача, не враховоючи обробку помилок, яку описано в розділі «Обробка помилок»

protected abstract List<List<InlineKeyboardButton>> getDefaultKeyboard(long chatId);

Метод для зміни стану бота, згідно з запитом переходу на інший стан бота.

protected BotApiMethod redirectFromCallback(CallbackQuery callbackQuery, Map<String, BotState> map)

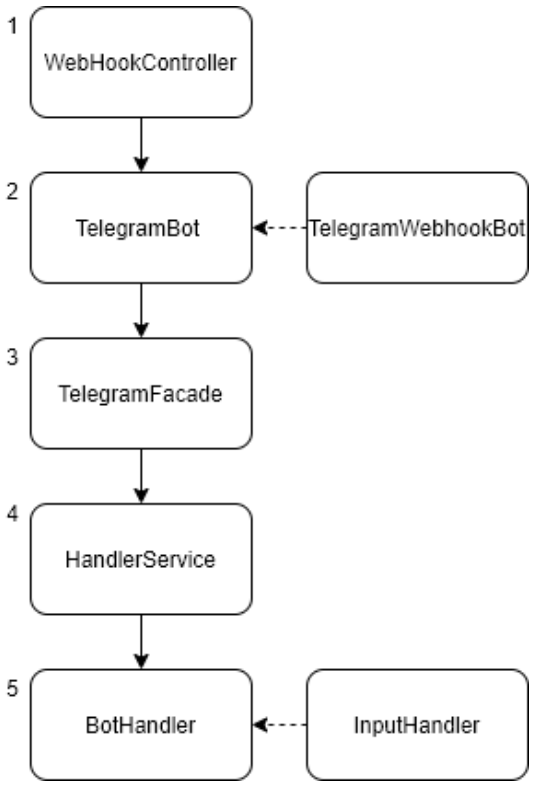

Рисунок 2.2 – Алгоритм обробки запитів від користувача

Не менш важлививими для всього процесу, є методи *getSendMessage* та *getEditMessage*, що виконують роль білдерів для процесу формування відповіді.

protected SendMessage getSendMessage(long chatId, String text, InlineKeyboardMarkup keyboard)

protected EditMessageText getEditMessage(long chatId, String text, InlineKeyboardMarkup keyboard, Message message)

Останні, але не меньш важливі є методи для роботи з сповіщеннями *notificationKeyboard* та *sendNotification*, які описані в розділі «Робота зі сповіщеннями»

2.2 Робота з даними

Для роботи з даними зазвичай використовуються три підходи:
- робота зі статичними даними
- робота з даними, які змінюються не часто
- робота з даними, які змінюються часто

Перший, направлений на збереження даних, взагалі не змінюються протягом роботи бота, До цього типу відносяться дані інтерфейсу, як назви кожного з пункту меню, зі збереженням локалізації для кожного пункту. Він може бути представленний в вигляді – файлів ресурів чи за допомогою, пар ключів значень, що буде зберігатись в кеші серверу постійно. Для використання ресурсів з локалізацією в *Java* та відповідно і в *Kotlin,* при створенні папки ресурсів за замовчуванням можна вказувати суфікс, що буде вказувати на певну мову, так для тексту меню можна створити файли з назвою *menuTitles*, для англійської мови, застосуємо суфікс "*_eu_EU*", для української - "*_ua_UA*", російської - "*_ru_RU*", ці файли буде об'єднано автоматично в *Resource Bundle* та доступ до них ми зможемо отримати через *MessageSource* об'єкт *bean,* який буде створено з вказанням де знаходяться папки ресурсів, а отримувати дані ресурсів з урахуванням локалізації, ми будемо через сервіс ресурсів представлений в боті, як *LocaleMessageService*. В такому випадку цей сервіс буде присутній у всіх обробниках подій, і за допомогою збережених для кожного користувача налаштувань мови, буде видано релевантну інформацію для користувача, на вибраній ним же мові з переліку доступних.

Для даних, які можуть змінюватись не часто, використовується спосіб зі збереженням в кеш програми. Спосіб побудування кешу може відрізнятись для

різних випадків. Для найменування категорій, товарів, ролей, мов і так далі, можна використати варіант з хеш таблицею, в яку будемо зберігати пари ключ значення, та ініціалізувати при повторному запуску додатку, з даних, які зберігаються в ресурсах чи в базі данних. Так як спосіб зі збереженням даних в базі даних, є найбільш зручним для збереження та доступу до них, постійне надсилання запитів на сервер де зберігається база даних, несе для клієнта затримки в отриманні контенту, тому бажано зменшувати час відгуку за рахунок кешування даних протогом певного часу. До даних, які бажано кешувати, відноситься об'єкт повідомлення користувача, згідно з побудованої стратегії, щоб редагувати об'єкт, його неохідно отримувати при першому віправленні повідомлення чи після поновлення роботи додатку, якщо перше повідомлення було видалене користувачем. Кожного разу при запиті від користувача, відбувається перевірка на наявність в кеші, даних про повідомлення користувача, якщо повідомлення нема, згідно з алгоримом, відбувається запит для отримання об'єкту повідомлення, що зберігається на сервері в серіалізованому вигляді в вигляді файлу, а також може бути представлений як текстове поле в базі даних, хоча цей спосіб, сеперечить концепції використання баз даних, інших варіантів нема, через не можливість створення нових об'єктів типу *Message,* що накладає обмеження для парсування даних, необхідних для відправлення редагованого повідомлення клієнту.

Зазвичай сесія користувача триває до десяти хвилин, що можна використати для оптимізації збережених в кеші ресурсів та організувати очищення даних, які не використовуються більше заданого часу. Для цього при запиті відбувається запуск відрахунку часу, щоб очистити дані про користувача. А також згідно з цим алгоримом, буде оновлено час, кожного разу при отриманні даних з кешу, щоб забезпечити безперервну інтеграцію кашу та бази даних в найбільш оптимізованому варіанті.

Іншим варіантом роботи з кешом є фреймворк *Redis*, який буде зберігати завчасно проініціалізовані дані в вигляді пари ключ значення та зберігатиметься в оперативній пам'яті серверу, що дасть змогу найбільш швидко отримувати дані, та не формувати рядки, що містять інформацію про товари. За замовчуванням, використовується підхід, діставання даних з бази даних та формування згідно з

методом побудови довідкової інформації кожного разу для товарів, проте, використовуючи цей спосіб, ми зможемо привязати кожен виклик бота для отримання інформації про конкретний товар до виклику, що оптимізує ресурси як і в попередньому випадку, проте працює на рівні викликів до оперативної пам'яті.

2.3 Робота з базою даних

Взаємодія з базою даних, містить ряд абстракцій, які дозволяють зменшити об'єм повторюваного коду, та спростити роботу над даними записування їх транзактивно. Для розуміння загального підходу до створення взаємодії з базою даних. Створюється сервіс, який містить інформацію про хост бази даних, логін, пароль користувача та через драйвер *JDBC* налаштовується з'єднання з ресурсом даних.

Для того щоб отримати дані, необхідно правильно сформувати запит в вигляді об'єкту *Statement*, який приймає як параметер *SQL* запит. Після чого, запит можна виконати потрібним методом *execute*

```
Statement statement = connection.createStatement();
String query = "SELECT * FROM table"
ResultSet resultSet = statement.executeQuery(query)
```

Зчитування даних так само полягає в послідовному зчитуванні потоку даних через *ResultSet*, які містяться в даних рядках бази даних. Таким чином, для того, щоб отримати дані про поле необхідно знати його тип та зчитувати саме той тип даних, який буде збережено на сервері, з урахуванням Наступною причиною, через яку спосіб «ручного» збирання даних є не ефективним – це обробка помилок отримання даних, спосіб повністю потребує описування роботи з помилками розробником, що не тільки додає роботи, а й зменшує читабельність коду, про що неоднократно заявляли, в вигляді фреймворіків, які спрощують роботу з помилками. Одним з яких є фреймворк *Lombok*, що дозволяє прокидати помилки на рівень обробки компілятором, іншим варіантом позбавитись від обробки помилок, пропонується мовою програмування, яка базується на *JVM – Kotlin*. В цій мові, робота з помилками не є обов'язковою і за замовчуваням, всі помилки прокидаються до найнижчого рівня, щоб бути обробленим компілятором та

продовжити роботу, якщо це можливо, або ж сповістити користувача та завершити роботу аварійно.

Для спрощення робот, з базами даних, використовується фреймворк *Spring Data JPA* з використання *Hibernate*, та через який робота з базою даних зводиться до написання коду на *Java*, і може виконуватись без написання запитів до бази даних на мові *SQL*. Такий підхід полягає в створенні класу позначеному анотацією *@Entity* та присвоєнні йому полів, з обовязковим полем id позначеним анотацією *@Id* та за бажанням анотацією, яка буде генерувати *id* самостійно, в противному випадку ми беремо на себе відповідальність за створення *id*.

```
@Entity
@Getter
@Setter
@FieldDefaults(level = AccessLevel.PRIVATE)
public class Privilege {
   @Id
   @GeneratedValue
   private Long id;
   private String field;
}
```

Використовуючи дану абстракцію при запуску додатку буде зчитано параметри вказані в application.properties файлі

```
spring.datasource.url = jdbc:mysql://localhost:3306/schema_db
spring.datasource.username=username
spring.datasource.password=****
spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver
```

У випадку, якщо за заданим адресом бази даних не буде знайдено необхідної таблиці чи поля в таблиці, яке вказано в класі сутностей, його буде створенно без участі розробника, та буде забезпечена можливість змінювати дані таблиці за допомогою звичайних методів ініціалізації полів, що є наступним рівнем абстракції над базою даних. Проте, досі нема можливості отримувати данні з таблиці. Для цього необхідно створити клас який розширюється інтерфейсом *CrudRepository* та

містить посилання на *Entity* клас, описаний вище. Так за допомогою написання шаблонних абстрактних методів буде створено методи, через які можна виконувати роботу з таблицями бази даних. При цьому буде за замовчуванням доступний перелік методів, таких як:

findById – знаходження об'єкту за обов'язковим полем *id*,

findAll – для отримання всіх даних з таблиці,

save – збереження об'єкту *Entity*, як рядка в таблиці,

saveAll – збереження колекції об'єктів в таблиці,

delete – видалення заданого об'єкту з таблиці,

deleteAll – видалення всіх об'єктів таблиці ,

count – кількість об'єктів,

existsById – перевірка наявності об'єкту в таблиці та інші.

Найпростіший *CrudRepository* містить лише назву інтерфейсу та виглядає просто, хоча виконує більшість необхідної функціональності та генерує функції необхідні для вазаємодії з ботом

@Repository

interface UserActionRepository extends CrudRepository< UserData, Long>

findBy – префікс, для пошуку за полем, яке необхідно вказати після нього в «верблюжому стилі», як приклад, *findByField*– функція яка виконує пошук першого співпадіння в таблиці за полем з назвою *field*.

За аналогією наведеною вище так само можна формувати запити для *delete*, *exist*. Також можна з'єднувати умови пошуку за допомогою ключових слів *And*, *Or*, та після яких буде вказано назву поля за яким теж необхідно перевірити на співпадіння значень переданих методу. Або ж накласти обмеження на існуючі поля за певним параметром за допомогою структури *GreaterThan* (більше за значення вказане в параметрах), *LessThan* (меньше), *LessThanEqual* (меньше або дорівнює значенню), *GreaterThanEqual* (більше або дорівнює). *Between* (знаходиться в діапазоні значень).

Таким чином можна створити складні запитии з багатьма фільтрами, як приклад можна привести метод, який буде повертати всі наявні поля *UserData* в

яких вік менший ніж у змінній, яку ми передаємо як параметр, або ж вік не вказаний взагалі.

Такий метод може мати застосування в випадках, коли необхідно вказати вікову групу покупців, щоб видавати релевантну інформацію, за інтересами цієї вікової групи.

List<UserData> findAllByIdAndAgeIsLessThanEqualOrAgeIsNull(int age);

Усі дії з даними всередині додатку згідно з *MVC* необхідно здійснювати всередині сервісів, роботи з даними, які будуть об’днані абстрактним класом.

2.4 Висновки до розділу

В розділі було розглянуто та описано спосіб розробки чат-боту базований на архітектурі серверно орієнтованого додатку з використанням *MVC*. Було порівняно способи роботи з «чутливими даними» всередині додатку та способи налаштування доступу до данних захищеним способом з використанням технологій шифрування та перевірки за допомогою хеш-кодів для файлів ресурсів, зберігання даних в ресурсах та їх локалізація. Описана взаємодія з кешом програми для обробки запитів, що не потребують оновлення інформації миттєво. Було описано методи взаємодії та формування таблиць баз даних, за допомогою фреймворку *Spring Data JPA* та використання *Hibernate* з *MySQL* на найнижчому рівні взаємодії з даними. Було описано архітектуру та алгоритми роботи додатку при запитах з *Telegram*

РОЗДІЛ 3. РЕЗУЛЬТАТ РОЗРОБКИ ЧАТ-БОТА

Програмне забезпечення дозволяє організувати більшу частину процесів повязаних з вибором товару, замовленням товару, обробкою замовлень, до якого входять: сплата чеків в онлайн сервісі, доставка та оцінка якості доставки товару.

Програмний комплекс складається з таких основних компонентів як: меню ресторану, корзина товарів, замовлення, та налаштування бота та профілю користувача. Для описання послідовності дій користувача при попаданні в бот, необхідно описати кожен модуль даного програмного забезпечення.

3.1 Стартовий стан додатку

Згідно з правилами телеграму боти не можуть відправляти повідомлення перш ніж користувач, не розпочне роботу в боті, для цього використовується стартова команда, яка сигналізує про початок роботи з користувачам. Ця команда містить данні, які необхідна для стартової ініціалізації даних про користувача, як ім’я, прізвище, мова додатку та інші дані. Проте головними даними для ідентифікації користувача, неодіхдно використовувати значення «ідентифікатор чату» (*chatId*), яке унікальне для кожно користувача, і для необхідне для кожно з запитів до боту. Так, маючи ці данні, ми зможемо відправляти в чат повідомлення, після ініціалізації.

Видаливши повідомлення користувача, ми формуємо колонку користувача в базі даних, та відправляємо повідомлення, яке буде далі основним полігоном для відображення додатку згідно з концептом «чату в одному повідомлені»

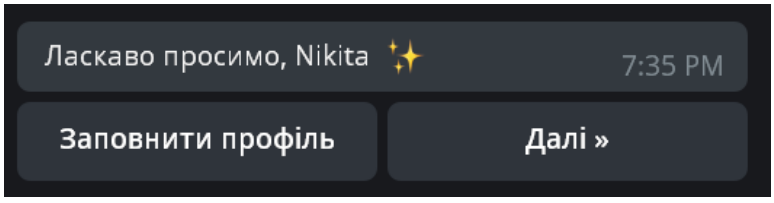

Рисунок 3.1 – Скріншот меню привітання від бота

3.2 Головне меню

Для користувачів які вперше ознайомлюються з чатом згідно з логікою, буде сховано більша частина голвоного меню, і з доступних варіантів залишаються «Меню» та «Налаштування». Побудування меню в чаті, базується на вбудованих в повідомлення кнопках, що описуються в методі *getDefaultMenu*, для кожного

обробника. «Головне меню», як стан позначається в перерахуванні *BotState*, та має значення «*MAIN_MENU*».

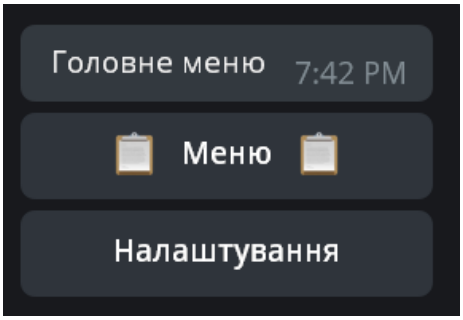

Рисунок 3.2 – Скріншот початкового стану головного меню

Після додавання в замовлення першого товару, в головному меню появляється, пункт в якому буде відображено, кількість позицій в замовлені.

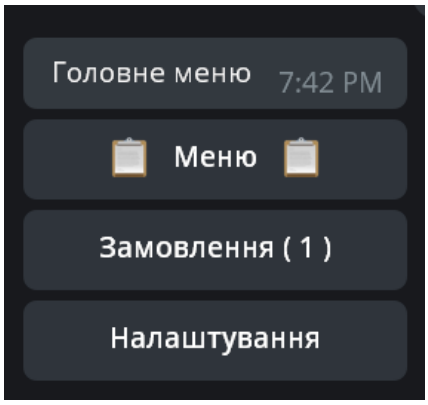

Рисунок 3.3 – Скріншот стану головного меню з товарами в замовленні

Після підтвердження замовлення, для відображення стає доступним пункт меню – Чеки, в якому зберігаються деталі замовлення, та цифра вказує на кількість нових замовлень в чеках

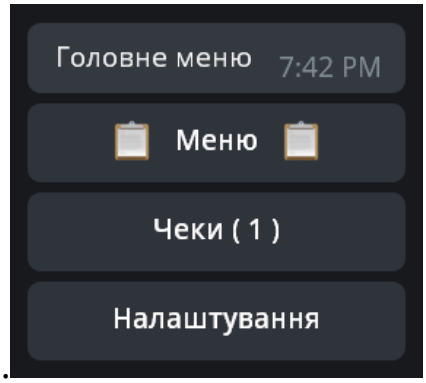

Рисунок 3.4 – Скріншот стану головного меню з замовленнями в обробці

Для адміністраторів, з головного меню доступна можливість перейти в пункт-меню «Сповіщення», в якому можна відправляти повідомлення клієнтам.

3.3 Меню ресторану

Як було зазначено в попередньому пункті, ми можемо через навігацію потрапити до меню, яке розбите на категорії, що попердньо зазначенні в програмі. В подальшій розробці реалізацію по додаванню категорій, необхідно зробити такми чином, щоб можна було додавти кожну з категорій, без втручання розробника через інтерфейс чату, як це реалізовано для товарів. Наразі, це неможливо, через те, що категорії товарів підтримують локалізацію категорій. Категорії, зберігаються в вгляді структури перерахування.

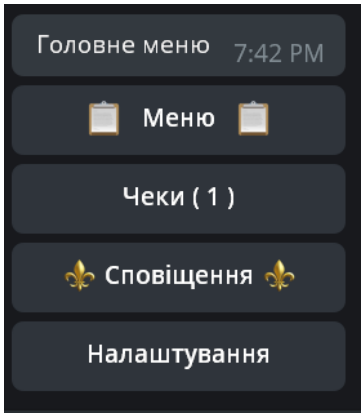

Рисунок 3.5 – Скріншот головного меню з пунктом «Сповіщення»

В даному меню, ми можемо бачити структуру, яка повязана, з даними які можна організувати у списки, та проходження по яким може бути здійснено за допомогою навігації, ака дозволяє, бачити категорії сторінками по п'ять пунктів. Якщо ж, пунктів в меню меньше а ніж це число. Навігація не буде відображатись і робота додатку не зміниться.

3.4 Навігація по списках даних

За навігацію по спискам товарів, відповідає абстрактний клас, що наслідується від базового класу обробників подій, тому може бути використаний для наслідування для обробників подій, в яких містяться дані, що можуть бути помічені як за допомогою абстрактного класу Data, який передаєтьяся через сервіси роботи з даними такими як *ProductDataService* (для обробки даних продуктів класу *ProductData*), який в свою чергу наслідується від *DataService*, що необхідно для навігації, адже за допомогою нього можна отримати всі данні в вигляді списку товарів, які можна буде в кінцевому варіанті розділити.

Головний клас обробки таких списків має назву *NavigationHandler* приймає в якості параметрів, окрім тих що вже описані для *InputHandler*, ще й параметр в вигляді довільного типу даних, який наслідується від *Data* інтерфейсу.

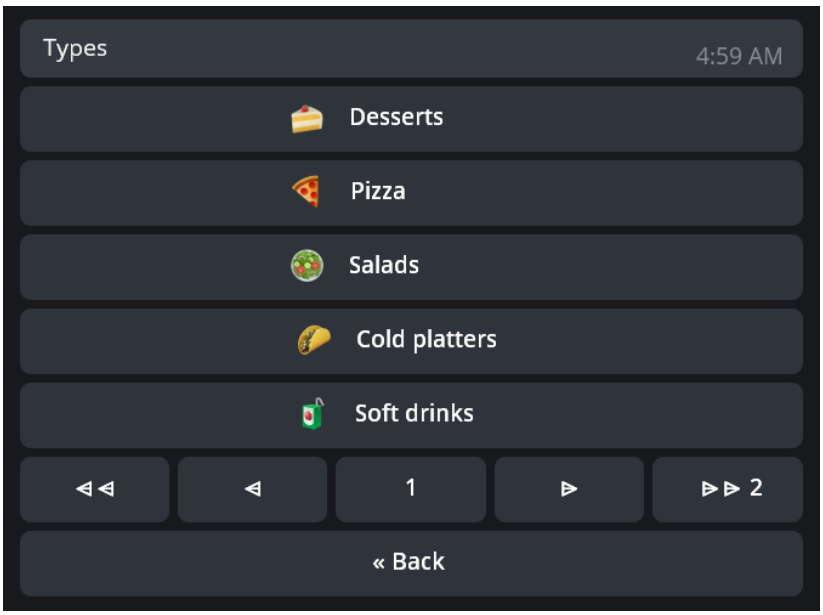

Рисунок 3.6 – Скріншот меню категорій товарів

Клас навігаційного обробника має такі методи та абстрактні поля:

Абстрактне поле, для отримання даних типу переданого як *Data* при створенні обробника подій за допомогою методу *getAll* в інтерфейсі *DataService*

abstract var dataService: DataService <E>

Метод *getPageList* для отримання списку даних в вигляді вбудованої клавіатури, з кастомізацію вигляду тексту, за допомогою методу *getText*, що передається в метод, як лямбда вираз, проте може бути замінений посиланням на метод класу, так як тут і далі використовується *Kotlin* мова, в якій кожен метод, може бути представлений в вигляді об’єкту, що надає можливість оперувати данами більш гнучко порівняно зі звичайними можливостями *Java*, яка стала підгрунтям *Kotlin*

fun getPageList(chatId: Long, getText: (E, Long) -> String, data: String,

actionType: ActionType,): MutableList<MutableList<InlineKeyboardButton>>

Метод *processNavigationCallback* призначений для обробки переходів між сторінками, за допомогою зворотніх сигналів, від вбудованої в повідомлення клавіатури навігації по спискам даних.

protected fun processNavigationCallback(callbackQuery: CallbackQuery, actionType: ActionType): BotApiMethod<>?*

Для зберігання даних, які не можуть бути передані всередині повідомлень, ми використовуємо окрему таблицю, яка містить статус лише три значення: chatId, ActionType, та значення яке передається як об’єкт, що дозволяє зберігати для користувача номер сторінки, на якій він перебуває,а також ввести систему в якій для кожного з користувачів, може бути лише один тип дії в якому він буде знаходитись. Для кожного типу навігації, дія буде інша, тому ми уникатимемо накладання та втрату інформації, що в кінцевому варіанті має надати більш стійкості до системи проти втрати даних.

Метод *getNavigationRow* повертає рядок вбудованої клавіатури

private fun getNavigationRow(current: Long, chatId: Long): MutableList<InlineKeyboardButton>

3.5 Меню списку товарів

Меню товарів відображає товри з категорії вибраної в меню типів товарів та працює на сонові принципу навігації, описаному в розділі «Навігація по спискам даних». Дане меню відображає список товарів відсортований за назвою, а також показує товари зі знижками з старою ціною та відповідно зі знижкою. Також має можливість додавання товарів до даної категорії, що є можливим для адміністраторів чат боту.

3.6 Сторінка товару

Сторінка товару описує товари за такими критеріями, як назва, ціна (зі знижкою, якщо є така) та опис товару, який містить доаткову інформацію про товар, наприклад склад, грамовку, та вказується опціонально.

Також це сторінка меню, дає можливість додавати товари до замовлення. Це можна зробити один з двох способів. Скористатись меню з позначкам «+» та «-» та довати по одному товару до замовлення, або ж скористатись командами, які дадуть можливість додавати великі кількості товарів, або ж видаляти зі списку, якщо це необхідно. Для додавання, необхідно в меню товару, відправити команду яка

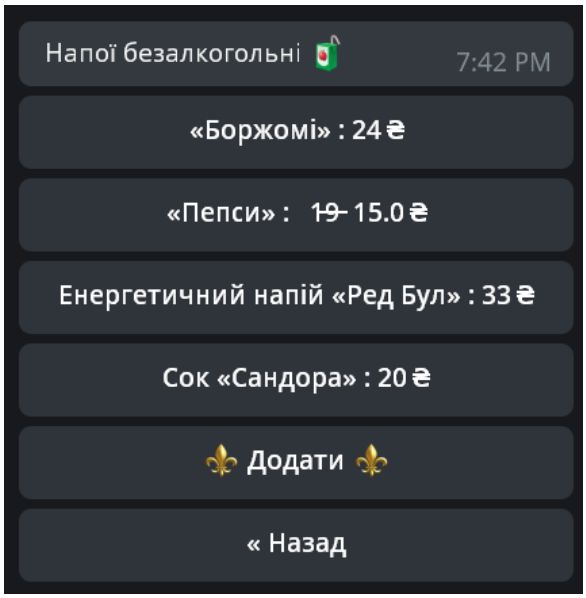

Рисунок 3.7 – Скріншот меню товарів з можливістю додавання нових товарів

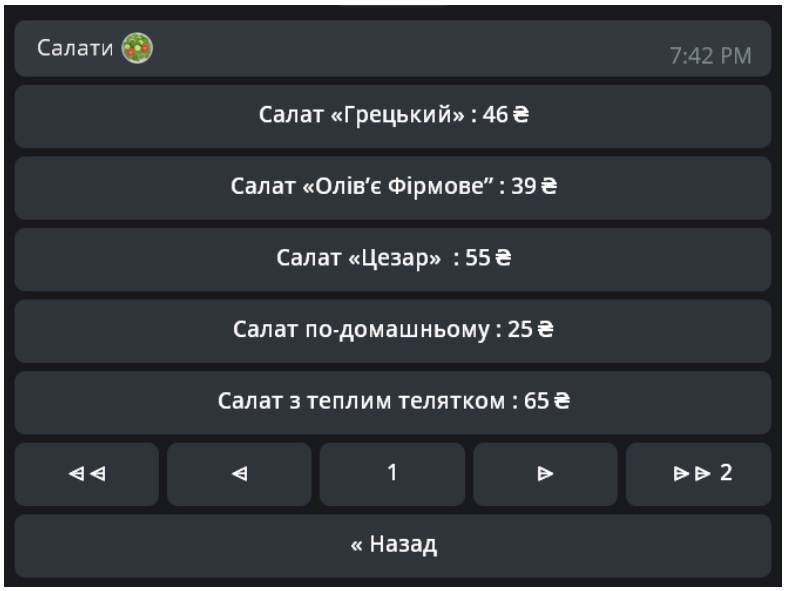

Рисунок 3.8 – Скріншот списку товарів з навігацією

матиме таку структуру «*/order_plus_ 5*», де замість цифри може бути будь яке числов межах цілочисленої змінної. Для зменшення кількості товарів в замовленні, можна використати команду, яка формується тик само ж, як і попередня, проте замість підрядка «*plus*» необхідно використати суфікс команди «*minus*», що зменшить число товарів на задане число таварів, проте не меньше нуля одиниць. Як приклад «*/order_minus_ 5*» при заданій кількості товарів в 4, зменшить число до 0, а не в -1.

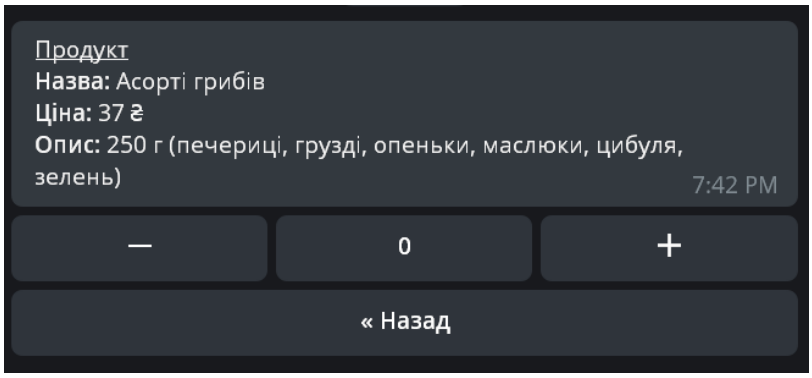

Рисунок 3.9 – Скріншот сторінки меню товару

Для адміністратора додатку доступні також такі функції, як видалення, редагування товарів, які позначаються як кнопки «Видалити» та «Редагувати»

відповідно.

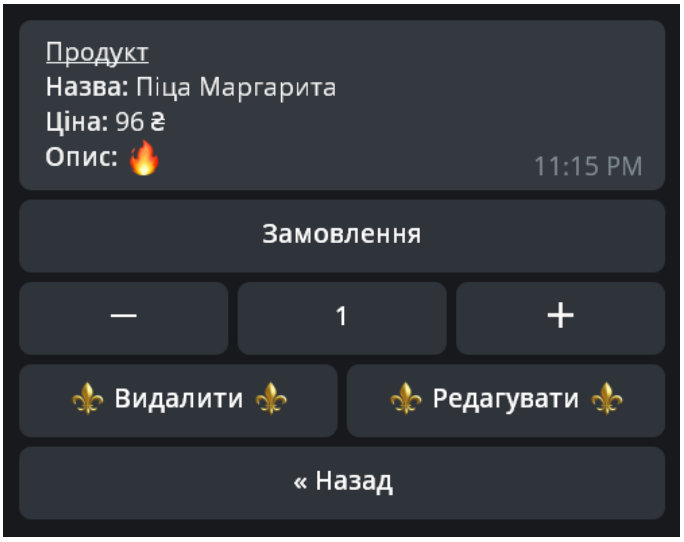

Рисунок 3.10 – Скріншот сторінки меню товару з можливостями адміністратора

Кнопка видалення – видаляє товар, та повертає до список товарів. Кнопка «Замовлення», перенаправляє в меню, де можна отримати список товарів доданих до замовлення.

3.7 Редагування товару

Зміна товару побудована на послідовному заповнені форми товару з обробкою відповідей від користувачей. Всі дії обробляються одним оброблювачем подій, який відповідає за редагування товару – *EditProduct*. При попаданні на метод *handler*, виконується зміна стану на початковий, якому відповідає відпправка запиту користувачу на введеня «назви» товару, після відправлення повідомлення у відповідь воно буде перевірене на валідність даних, які ввів користувач. Після

отримання валідних даних, користувач отримує запит на введення настпного поля – ціни товару.

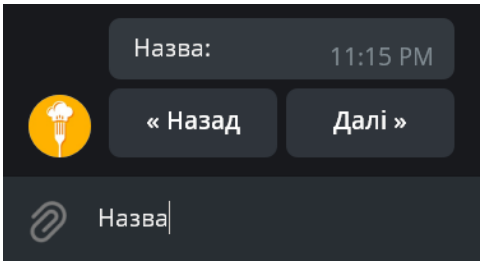

Рисунок 3.11 – Скріншот заповнення форми для поля «назва» товару

3.8 Обробка помилок вводу

Обробка помилок вводу користувача, виконується за допомогою додаткового параметру, який відправляється для формування повідомлення і додається на етапі перевірки на наявність тегу помилки. Всі помилки додаються, якщо не введені дані не відповідають умові, що зазвичай формується в вигляді, регулярного виразу чи в вигляді звичайного булевого виразу. Наприклад для даних, що за замовчуванням має значення 0, та не може бути від'ємним –увипадку, якщо буде вказане від'ємне число, буде отримано повідомлення помилки, яке містить деталі даної помилки. Всі помилки несуть лише інформативну властивість та не перенаправляють на інші стани, при цьому для коректної роботи введення форми, не відбувається перехід до наступного стану, доки корстувач не вибере пункт «Далі» або «Назад» чи введе валідні дані.

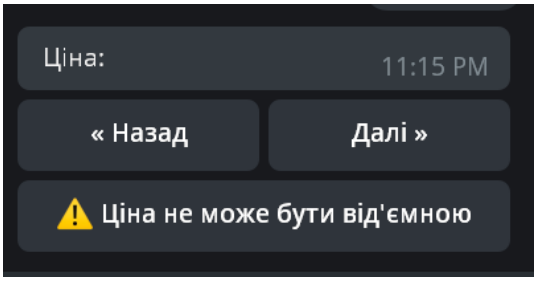

Рисунок 3.12 – Повідомлення про помилку

3.9 Меню замовлення

Після додавання товарів до замовлення в нас є можливість переглянути товари в замовленні та якщо нас не влаштовує ціна, кількість чи взагалі ми хочемо прибрати товар зі списку замовлення, ми можемо повернутись до меню товару скориставшись кнопками товару.

Якщо клієнт визначився, що замовлення йому підходить і він готовий перейти

до оплати, ми можемо створити чек і перейти до його обробки всередині системи іншим користувачем, який має права робітника закладу.

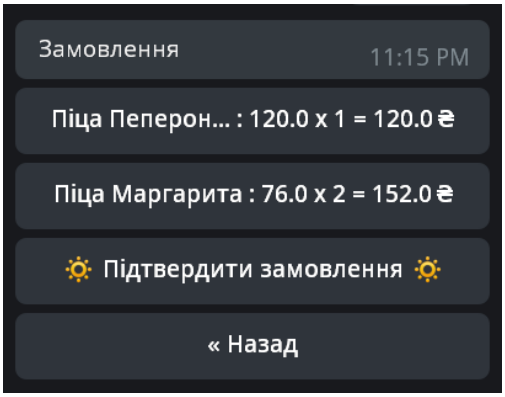

Рисунок 3.13 – Скріншот меню замовлення

3.10 Меню обробки замовлення

Обробка замовлення є послідовним процесом, що починається з прийняття замовлення робітником і до етапу оцінення якості. Першим етапом в обробці замовлення є підтвердження можливості прийняття замовлення в обробку. Для подальшої обробки замовлення, необхідно щоб було вказано кінцеву адресу доставки, що можна зробити за допомогою відправки локації вручну, якщо необхідно відправити замовлення на адресу, що відрізняється від вашої поточної або ж вам потрібно змінити адресу доставки. Для пересічного користувача, достатньо буде лише відправити команду «*/location*», після чого користувач отримає звернення, для надсилання поточної геолокації. Після надсилання геолокації, робітник ресторану може змінити статус замовлення на «Оплата товару».

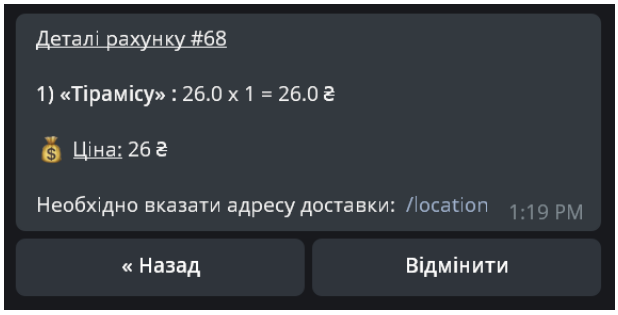

Рисунок 3.14 – Скріншот меню підтвердження замовлення

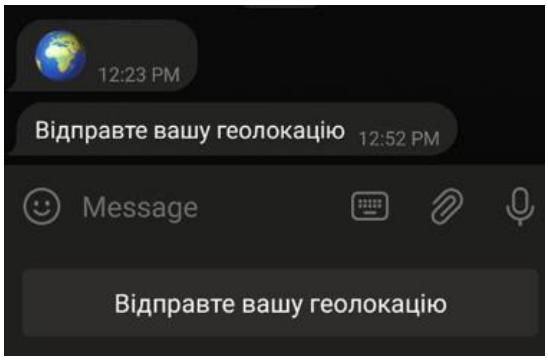

Рисунок 3.15 – Скріншот запиту геолокації

Оплата замовлення може відбуватись двома способами, за допомогою готівки, при отриманні товару, та за допомогою сервісу *LiqPay*, який дозволяє здійснювати оплату в інтернеті та отримувати дані про сплату суми за товар.

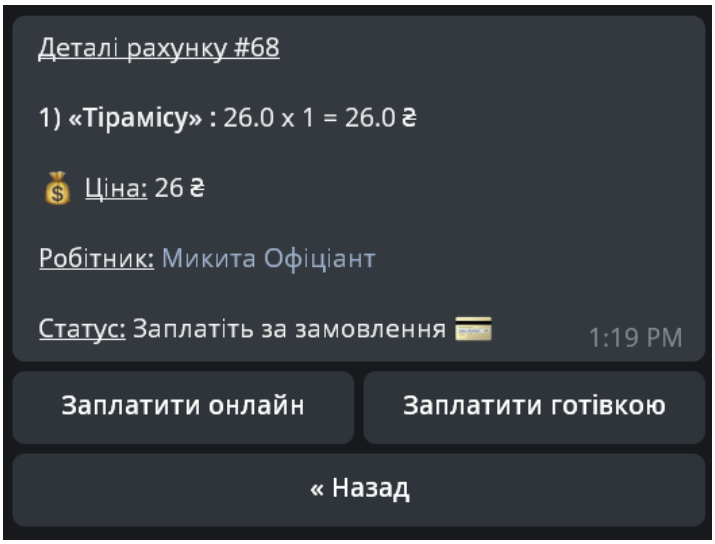

Рисунок 3.16 – Скріншот меню оплати замовлення

Перший варіант, встановлює «Вид оплати» як готівковий. Інший, формує посилання на запит оплати послуг на сервісі *LiqPay* за допомогою *API* представленого розробниками сервісу.

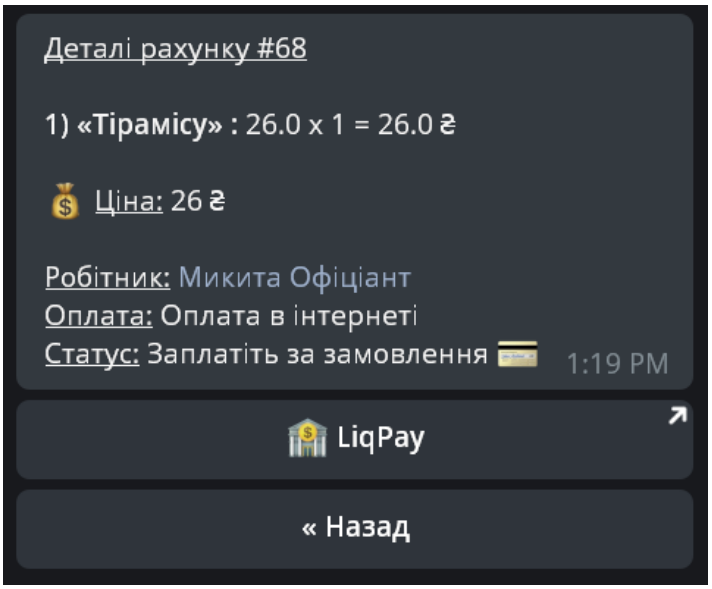

Рисунок 3.17 – Скріншот меню оплати в інтернеті

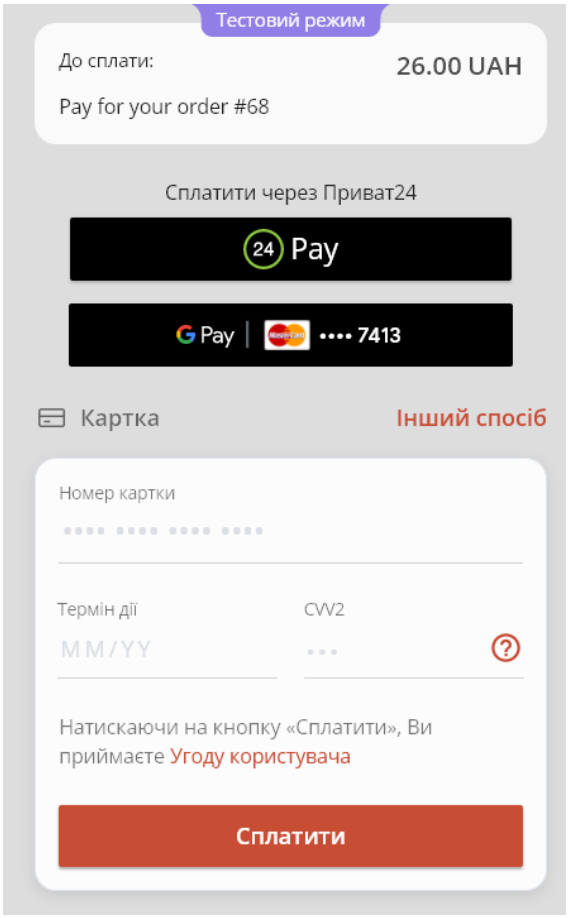

Рисунок 3.18 – Скріншот вікна оплати LiqPay

Після оплати замовлення одним з доступних методів : *Google Pay, Privat24* чи за допомогою реквізитів карти, клієнта буде автоматично перенаправлено назад до клієнту *Telegram*, при цьому буде очікуватись запит на оплату послуги від сервісу. Після отримання коштів на рахунок, буде автоматично змінено стан замовлення на «В процесі», який відповідає за приготування, спакування. Останій

підетап процесу приготування - видача людині, яка проводить доставку товарів та переведення в стан – «Доставка товару».

Доставка товару здійснюєть за адресом вказаним при створенні замовлення, та віддається людині у вигляді кнопки посилання на *Google Maps* додаток з вказанням адреси доставки клієнта.

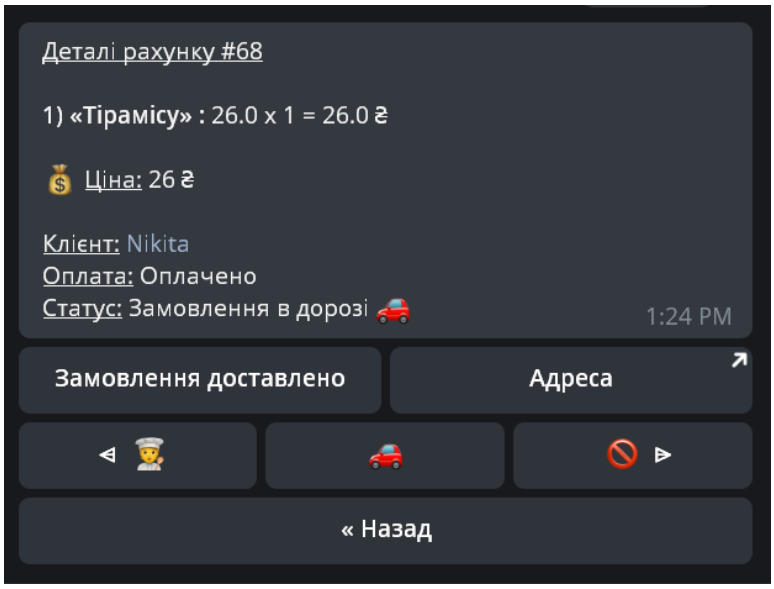

Рисунок 3.19 – Скріншот меню доставки товару

Запит до *Google Maps* формується на основі посилання пошуку, що формує запит за вказаними довготою і широтою. І в результаті, для людини яка доставляє залишається виконати такі кроки, як отримати адрес, довести замовлення за адресом. Після чого необхідно підтвердити доставку замовлення за допомогою вбудованої кнопки «Замовлення доставлено» та очікувати на реакцію клієнта. Для того щоб швидше взаємодіяти з користувачем, було добавлені: можливість написати клієнту через посилання на його профіль або ж набрати за допомогою номеру, якщо він є вказаний. І як альтернативний варіант оповіщення клієнта було зроблено систему оповіщень про зміну стану замовлення, яке буде описане в главі «Сповіщення».

Після сповіщення клієнта у нього є два вибори для підтвердження доставки замовлення або ж відправлення сповіщення людині, яка доставляє про те що замовлення не прийнято. Якщо ж було підтверджено, що замовлення отримано, тоді клієнту буде запропоновано оцінити сервіс, за допомогою рестингу, який буде позначатись від 1 до 5.

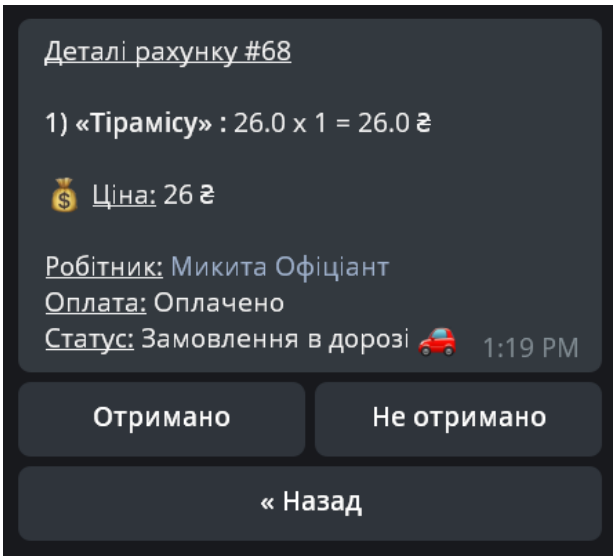

Рисунок 3.20– Скріншот меню отримання замовлення

Якщо клієнтом було виставлено мінімальну оцінку сервісу, тоді для уточнення деталей буде відправлено сповіщення для людини, яка відповідає за обробку жалоб клієнтів. Таким чином буде отримано зворотній зв’язок від клієнта, та здійснено виправлення програмного продукту, якщо проблеми виникали у зв’язку з роботою бота, чи оброблено заявку, повязану з якістю послуг.

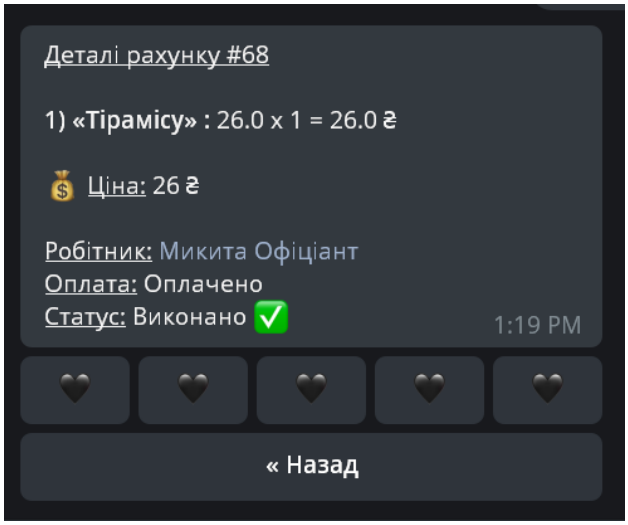

Рисунок 3.21 – Скріншот меню оцінення доставки

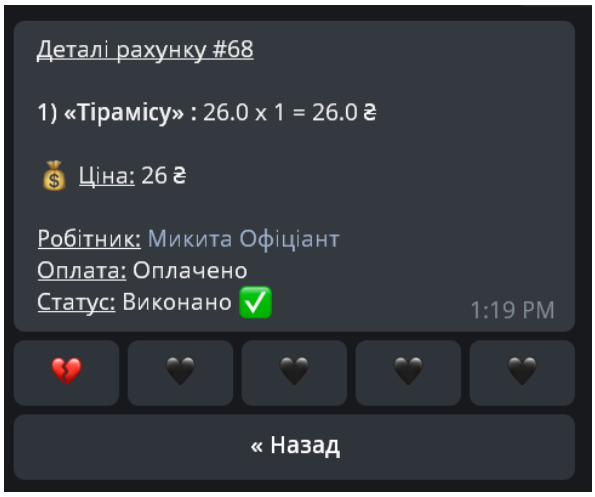

Рисунок 3.22– Скріншот меню оцінення доставки з значенням «1»

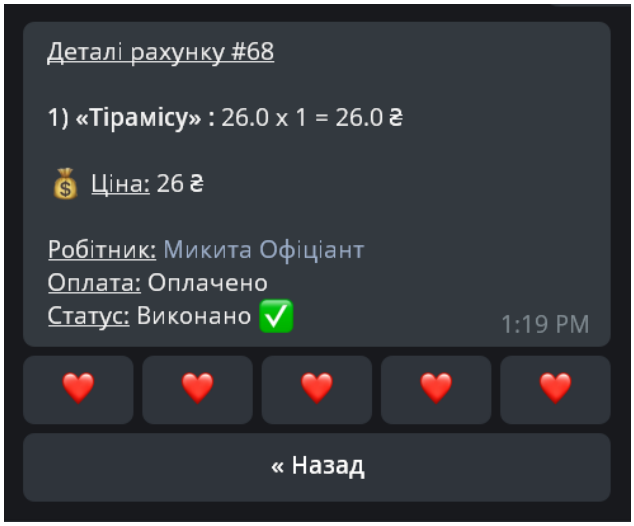

Рисунок 3.23– Скріншот меню оцінення доставки з значенням «5»

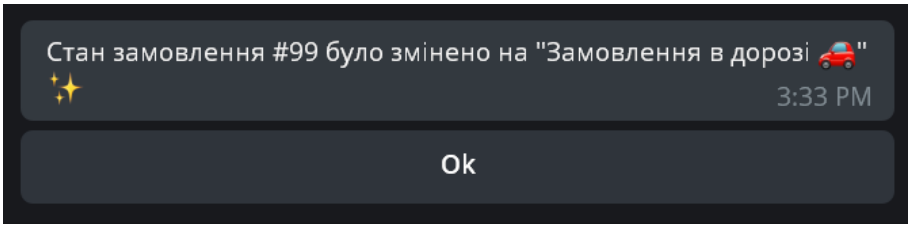

Рисунок 3.24– Скріншот сповіщення про зміну стану

3.12 Список меню чеків

Список чеків побудований на основі описаного в розділі «Навігація по списках даних» інтерфейсу *NavigationHandler* та містить коротку інформацію про чеки замовлень, а саме – номер замовлення, стан замовлення та ціну замовлення. Стану замовлення відповідає *Emoji*, яке описує, що саме відбувається в замовленні:

Піктограма стану	Назва стану
	Оплата замовлення
	Нове замовлення
	В процесі приготування
	Доставка товару
	Замовлення виконано

Таблиця 3.1 – Позначення станів за допомогою піктограм

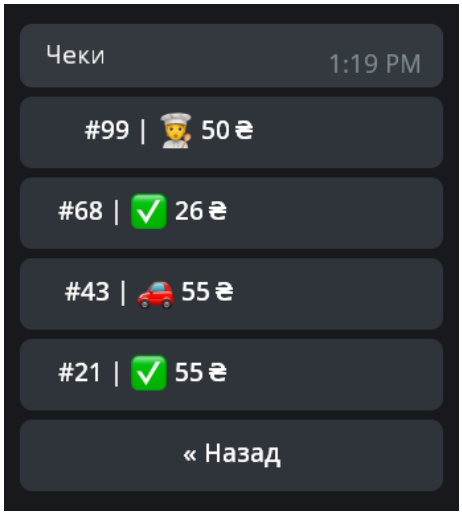

Рисунок 3.25 – Скріншот меню списку чеків

3.13 Сповіщення

В головному меню пункт «Сповіщення» відповідає за розсилання сповіщень користувачам заданої категорії. Дана функція доступна лише адміністарторам чат-бота. Меню вибору категорії типу повідомлення має на вибір лише текстовий тип данних, проте є оптимізовний для додавання інших типів сповіщень в майбутніх доробках чат-бота.

Після вибору типу повідомлення, клієнт переходить в стан введення тексту сповіщення. Далі після введення повідомлення сповіщення необхідно вибрати категорію користувачів, яка отримає сповіщення. Це може бути корисно, коли необхідно відправити повідомлення акційне для користувачів, при цьому не сповіщаючи робіників, для яких чат є робочим простором.

Після натискання кнопки «Ok» сповіщення буде видалено, і при цьому бот залишатиметься в концепції одного повідомлення. Це досягається завдяки обробки зворотнього сигналу від користувача, що вказує на те що повідомлення необхідно видалити. Сигнал оброблюється класах які наслідуються від *InputHandler*.

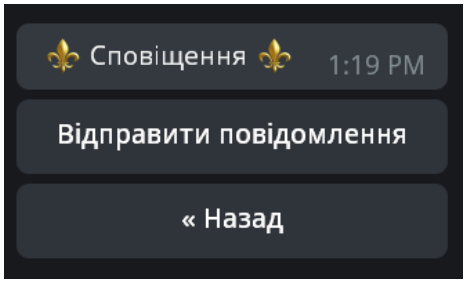

Рисунок 3.26 – Скріншот меню відправки повідомлень

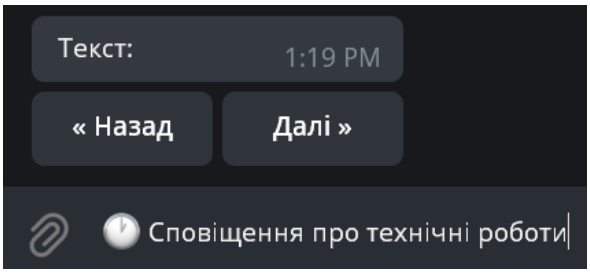

Рисунок 3.27 – Скріншот меню введення значення для тексту сповіщення

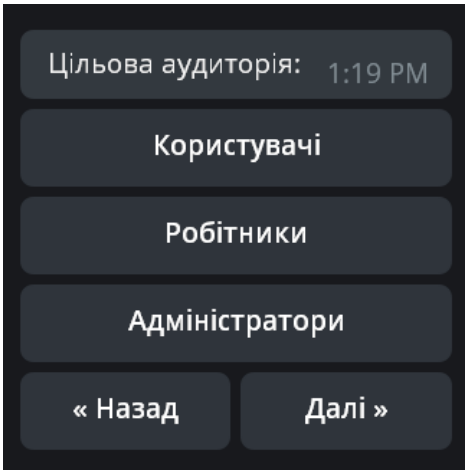

Рисунок 3.28 – Скріншот меню вибору категорії користувачів

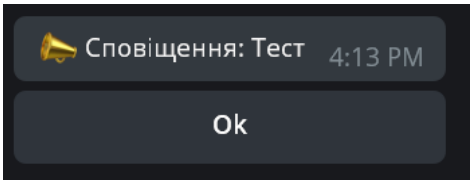

Рисунок 3.29 – Скріншот сповіщення для адміністраторів

3.14 Меню налаштування

Меню налаштування містить пункти меню, які відповідають за мову бота для кожного з користувачів, профіль користувача, налаштування прав користувачів та пункт підтримки користувачів.

3.15 Профіль користувача

Сторінка користувача, містить інформацію про користувача та дозволяє відображати лише заповнену інформацію. Містить інформацію про ім’я, прізвище користувача, його роль в додатку, електрону пошту для зв’язку в разі необхідності, стать клієнта. Будь який з цих пунктів форми може бути не вказаний, проте при заповнені профілю обов’язковим є пункт номер телефону для формування списку реальних користувачів, адже в мабутніх релізах додатку планується ввести купони, які будуть генеруватись для кожного з акаунтів, і реальність користувачів є одним з ключових моментів для надання знижок клієнтам.

Заповнення профілю відбувається схожим чином, як організовано для створення нових продуктів в меню категорій, проте містить в собі запити на інформацію про номер телефону, що викликає нехідність по особливому виконувати запити, щоб залишитись в концепції. Також було використано варіант опитування з завчасно зазначеним переліком відповідей, що необхідно обмежити вибір користувача до заданих варіантів та полегшити роботу з вводом даних у вигляді тексту.

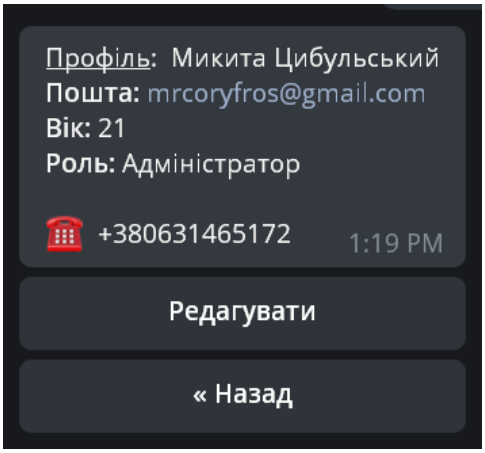

Рисунок 3.30 – Скріншот меню профілю користувача

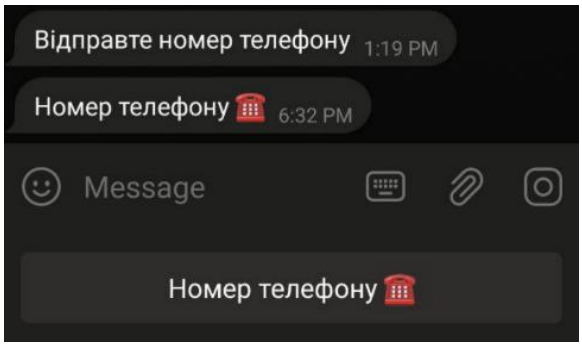

Рисунок 3.31 - Скріншот запиту на отримання телефоного номеру

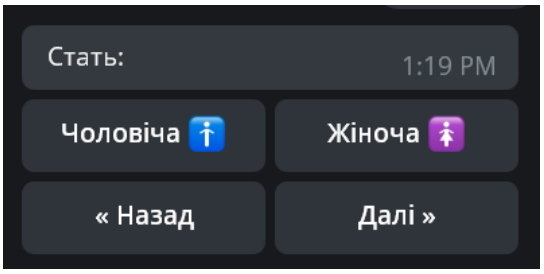

Рисунок 3.32 - Скріншот запиту на введення статі користувача

3.16 Меню вибору мови

Як було згадано в розділі про зберігання даних, дані інтерфейсу виключаючи назви продуктів є локалізованими та зберігаються в ресурсах. Тому є можливість налаштувати мову для конкретної людини, при цьому можливість додавати нові мови не є обмеженою, тому для нових мов необхідно буде описати вже існуючі теги, які привязуються до станів та назв в інтерфейсі програмі. Вибрана мова буде позначатись галочкою. На разі з доступних мов є російська, англійська, українська.

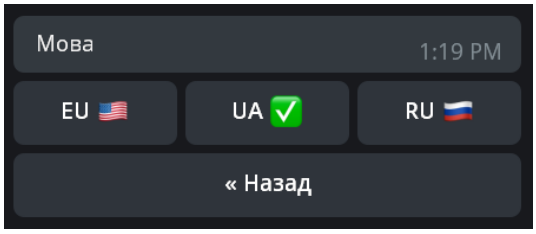

Рисунок 3.33 – Скріншот меню вибору мови

3.17 Меню управління ролями

Для розуміння необхідно описати ролі, та як з ними відбувається взаємодія з ними. Для початку в додатку існують бази даних які зберігають ключі кодної з ролей, та можливості які відповідають їм. За замовчуванням в кожного ж ролей є перелік звичайних можливостей і для створення доступу до нових можна їх зазначити в класі ініціалізації, що відповідає за те, щоб база даних при початковому запуску або під час деплою на сервери мала інформацію про ролі користувачів. При

початковому стані користувачів, необхідно вказати адміністратора вручну, а для інших за замовчуванням буде присвоєно права користувача.

Зміна ролей відбувається через меню в якому в залежності від ролі будуть відсортовані всі користувачі додатку та можна буде переглянути інформацію стосовно кожного з них.

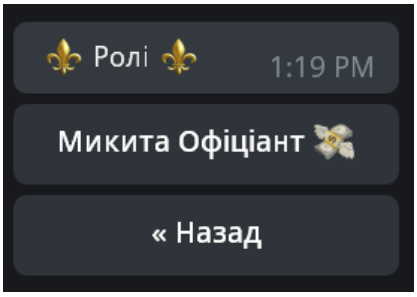

Рисунок 3.34 – Скріншот меню вибору користувача

Меню вибору ролі користувача дає можливість змінити роль лише до рівня, на якому знаходиться користувач. Якщо робітник може вибрати лише користувачів, які нижчі нього за рангом – та присвоїти йому права робітника. Для адміністраторів існує така ж сама система, яка дає можливість змінити робітнику права до свого рівня відповідальності. Проте для кожного з адміністраторів існує роль власника чату бота, який може змінити роль іншого адміністратора при цьому інші адміністратори, не можуть редагувати користувачів, які мають такі ж права, що і у них.

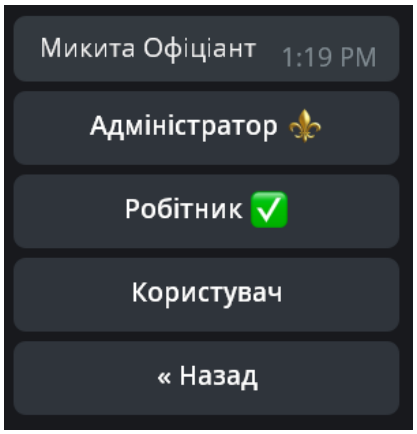

Рисунок 3.35 – Скріншот меню вибору ролі користувача

3.18 Сторінка підтримки

Сторінка підтримки містить інформацію про електрону пошту ресторану та номер телефону, та виконує функцію резервоного каналу зв’яку в разі нестправностей в

обробці, оплаті товарів та інших пробоем по’язаних з бізнес процесами, таких як пропозиції щодо вдосконалення бота, документація багів та інші.

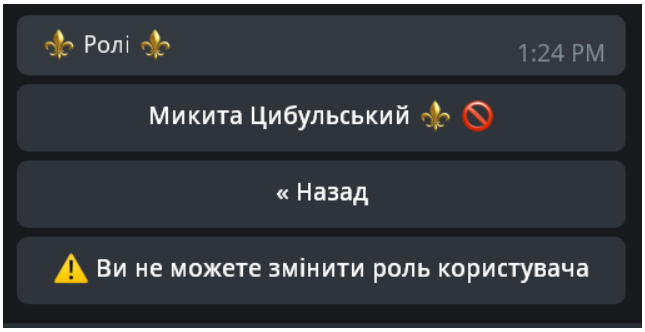

Рисунок 3.36 – Скріншот помилки зміни ролі користувача

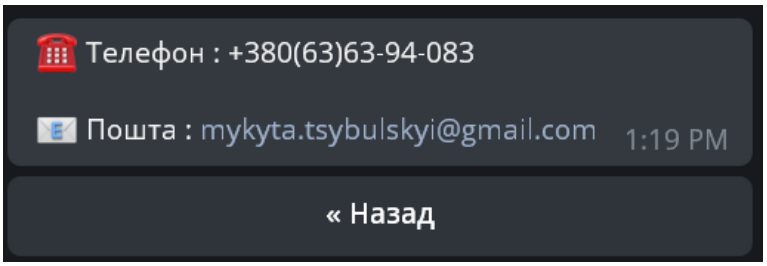

Рисунок 3.37 - Скріншот контактів підтримки

ВИСНОВКИ

В процесі розробки було оброблено дані аналогів чат-ботів, виведено характерні особливостей кожного з типів, прописано особливості побудування та маштабування рішень. Обговорено методи оптимізації роботи запитів та питання безпеки даних, які зберігаються локально так і в базах даних, в результаті чого було запропоновано рішення пробеми з використання трьохсторонньої переврки даних з урахуванням всіх особливостей запитів з якими може зіткнутись розробник при проектуванні та роботі з цими даними.

Під час проектування та розробки програмного додатку для вазаємодії з веб-інтерфейсом управління запитами від клієнта телеграму - *Telegram Bot API*. Було проведено дослідницьку роботу по створенню та вдосконалення уже існуючих методів взаємодії з кориистувачами чат-ботів. Було отримано в резльтаті концептуальний варіант вазаємодії з ботом, що базується на одному повідомленні і на відміну від інших типів взаємодії представляє собі чат-бота у вигляді веб-застосунку, всі дії якого відбуваються всеседині одного повідомлення не виходячи за рамки концепту окрім випадків, де це потребує логіка роботи або ж є обмеження зі сторони клієнта *Telegram* на відправлення чи отримання даних звичним способом, в даному випадку - «методу редагування повідомлення». На практиці було застосвоано знання з побудування *MVC* орієнтованих серверних додатків, що дозволило сформувати додаток орієнтований для внесення змін. Було зібрано та оброблено інформацію про процеси ресторанного бізнесу та їх автоматизацію на прикладі процесу доставки та обробки замовлень.

Використано стани для навігації в чаті, проведено роботу над універсальним поєднуванням схожих за базовими ознаками різноманітних сторінок та меню в різні групи об’єднані один інтерфейсом.

Було використано сервіси оплати за допомогою програмної взаємодії. Налаштовано сервіс для роботи з рахунками ФОП. Автоматизовано обробку оплати замовлення

Було досягнуто результату та проведено тестування додатку на групі людей залучених в бізнес систему, для оцінки та ознайомлення з новим продуктом, що вимаєгає подальшої роботи та вдосконалення. Виправлено рішення та внесено

покращення для архітектурних рішень, враховано бізнес-запити, які виникали в процесі розробки додатку

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ