Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ TELEGRAM-БОТУ ДЛЯ БРОНЮВАННЯ ПОСЛУГ ІЗ ВИКОРИСТАННЯМ NESTJS ТА AWS СЕРВІСІВ

DEVELOPMENT AND RESEARCH TELEGRAM BOT FOR BOOKING SERVICES USING NESTJS AND AWS SERVICES

спеціальність 121 Інженерія програмного забезпечення
освітня програма «Інженерії програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗм-21
Шведа В. О.

Керівник:
к.т.н., доцент
Повстяна Ю. С.

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2025 р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.
_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти: *магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: *«Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«__» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Шведі Валентину Олександровичу

1. Тема кваліфікаційної роботи: *Розробка та дослідження Telegram-боту для бронювання послуг із використанням NestJS та AWS сервісів*

Керівник роботи: *Повстяна Ю.С. к.т.н., доцент*

затверджені наказом закладу вищої освіти від «29» *березня 2025 р. № 190/01-02*

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«04» грудня 2025 р.*

3. Вихідні дані до роботи *технічне та програмне забезпечення ЕОМ*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *:аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проектування, опис функціонального наповнення об'єкта проектування, розробка й обґрунтування системного наповнення, оцінка ергономічних та надійністних параметрів проектованої системи.*

5. Перелік графічного матеріалу *10 рисунків, 7 лістингів коду.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Повстяна Ю. С.*		
Теоретичне дослідження та практична реалізація	*Повстяна Ю. С.*		
Експериментальне дослідження системи	*Повстяна Ю. С.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Андрущак І. Є.*		
Показник запозичень тексту	___%		
Академічна доброчесність	*Повстяна Ю. С.*		

7. Дата видачі завдання 02.04.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Провести огляд літературних джерел по темі кваліфікаційної роботи	02.05.2025	
2	Провести аналіз загальної проблеми і вибір напрямків дослідження	24.09.2025	
3	Розробити функціональну схему роботи програмного продукту	01.11.2025	
4	Описати засоби розробки об'єкта проектування	19.11.2025	
5	Практична реалізація об'єкта проектування	26.11.2025	
6	Розробити методику для проведення експерименту	05.11.2025	
7	Провести аналіз результатів експерименту	15.11.2025	
8	Здача чистового варіанту кваліфікаційної роботи на кафедру	04.12.2025	

Здобувач вищої освіти _____ Шведа В. О .

Керівник кваліфікаційної роботи _____ Повстяна Ю. С.

АНОТАЦІЯ

Шведа В. О. Розробка та дослідження Telegram-боту для бронювання послуг із використанням NestJS та AWS сервісів. Рукопис.

Кваліфікаційна робота магістра освітньої програми «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається зі вступу, трьох розділів, висновків, списку використаних джерел. У кваліфікаційній роботі розглядається проблема автоматизації процесів бронювання послуг із використанням чат-ботів у середовищі месенджера Telegram. Робота включає аналіз сучасних наукових досліджень і практичних рішень у сфері автоматизації сервісів, обґрунтування вибору технологічного стеку, проєктування архітектури системи та реалізацію Telegram-боту на основі фреймворку NestJS і хмарних сервісів Amazon Web Services.

У першому розділі роботи проаналізовано сучасний стан проблеми автоматизації бронювання послуг, розглянуто існуючі підходи до створення чат-ботів, їх архітектурні рішення та світові тенденції використання хмарних технологій. У другому розділі обґрунтовано вибір технологій, описано архітектуру Telegram-боту, модель бази даних та процес практичної реалізації системи з використанням NestJS, Telegraf, TypeORM і сервісів AWS. У третьому розділі проведено експериментальне дослідження роботи Telegram-боту, виконано тестування основних функціональних сценаріїв та проаналізовано результати, які підтверджують коректність, надійність і ефективність розробленого рішення.

Ключові слова: Telegram-бот, бронювання послуг, чат-боти, NestJS, AWS, безсерверна архітектура, реляційна база даних, хмарні сервіси.

ABSTRACT

Shveda V. O. Development and Research of a Telegram Bot for Booking Services Using NestJS and AWS Services. Manuscript.

Master's Thesis in the specialty Software Engineering. Lutsk National Technical University, Lutsk, 2025.

The master's thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices. The thesis addresses the problem of developing a Telegram bot for service booking using modern cloud and serverless technologies. The work includes a theoretical analysis of existing approaches to service booking automation, justification of the selected technology stack, system architecture design, and practical implementation of a scalable booking solution.

The first chapter analyzes existing scientific studies and practical solutions in the field of chatbot-based service automation, identifies their limitations, and outlines current trends in the use of cloud and serverless architectures. The second chapter focuses on the design and implementation of the Telegram bot, including data modeling, multi-step dialogue management, integration with AWS Lambda, API Gateway, and Amazon RDS, as well as the use of TypeORM for database interaction. The third chapter presents an experimental study of the developed system, including functional testing of booking scenarios and analysis of database records, confirming the correctness, stability, and efficiency of the proposed solution.

The results of the research demonstrate that the developed Telegram bot provides a reliable, scalable, and cost-effective solution for automating service booking processes and can be adapted for use in various business domains.

Keywords: Telegram bot, service booking, cloud computing, serverless architecture, NestJS, AWS, TypeORM, relational database.

ЗМІСТ

ВСТУП

У сучасних умовах цифровізації бізнес-процесів особливої актуальності набувають рішення, спрямовані на автоматизацію взаємодії між надавачами послуг і клієнтами. Одним із таких напрямів є автоматизація процесів бронювання послуг, яка дозволяє зменшити навантаження на персонал, підвищити оперативність обслуговування та забезпечити зручність для користувачів. На сьогодні широко застосовуються вебзастосунки та мобільні платформи для реалізації функцій бронювання, однак вони часто потребують окремої реєстрації, встановлення додаткового програмного забезпечення та значних витрат на розробку й підтримку.

Аналіз сучасних наукових досліджень і практичних рішень показує, що чат-боти, зокрема на базі месенджерів, дедалі частіше використовуються як ефективний інструмент автоматизації сервісних процесів. Telegram-боти вирізняються високою доступністю, простотою інтеграції та підтримкою інтерактивних сценаріїв взаємодії з користувачем. Водночас значна частина існуючих рішень орієнтована на вузькі сценарії використання, не забезпечує масштабованості для кількох бізнесів або не враховує можливості сучасних хмарних і безсерверних архітектур. Це свідчить про наявність прогалин у практичній реалізації універсальних, масштабованих та економічно ефективних систем бронювання на основі чат-ботів.

Світові тенденції розвитку інформаційних систем у сфері обслуговування клієнтів спрямовані на використання хмарних технологій, безсерверних обчислень та мікросервісної архітектури. Застосування платформ типу Amazon Web Services дозволяє забезпечити автоматичне масштабування, високу доступність і зменшення витрат на інфраструктуру. Поєднання чат-ботів із безсерверною архітектурою відкриває нові можливості для створення гнучких і масштабованих сервісів бронювання, що відповідають сучасним вимогам ринку.

Актуальність даної кваліфікаційної роботи зумовлена потребою у створенні ефективного програмного рішення для автоматизації бронювання послуг із використанням Telegram-боту, інтегрованого з хмарною безсерверною інфраструктурою. Запропонований підхід дозволяє спростити процес взаємодії з користувачами, підвищити надійність системи та зменшити витрати на її розгортання й обслуговування, що є важливим для малого та середнього бізнесу.

Метою кваліфікаційної роботи є розробка та дослідження Telegram-боту для бронювання послуг із використанням фреймворку NestJS та хмарних сервісів Amazon Web Services, а також експериментальна перевірка ефективності його функціонування.

Для досягнення поставленої мети в роботі передбачено виконання таких завдань:

– проаналізувати сучасні наукові дослідження та існуючі програмні рішення у сфері автоматизації бронювання послуг із використанням чат-ботів та хмарних технологій;

– обґрунтувати вибір технологій, фреймворків і хмарних сервісів для реалізації Telegram-боту бронювання послуг;

– спроєктувати структуру даних та розробити модель реляційної бази даних для зберігання інформації про бізнеси, послуги, часові інтервали та бронювання;

– розробити Telegram-бот для бронювання послуг із реалізацією багатоетапного сценарію взаємодії користувача та підтримкою нагадувань. Інтегрувати розроблений Telegram-бот у безсерверну хмарну інфраструктуру Amazon Web Services;

– провести експериментальне тестування розробленого Telegram-боту та проаналізувати результати його функціонування.

Об’єктом дослідження є процеси автоматизації бронювання послуг у інформаційних системах.

Предметом дослідження є методи, моделі та програмні засоби розробки Telegram-ботів для бронювання послуг із використанням безсерверної хмарної архітектури.

Наукова новизна роботи полягає у розробці та дослідженні підходу до реалізації багатокористувацького Telegram-боту для бронювання послуг на основі безсерверної архітектури Amazon Web Services із використанням механізмів керування станом діалогу та узгодженого управління ресурсами в реляційній базі даних.

Практична цінність роботи полягає у можливості використання розробленого Telegram-боту як готового програмного рішення для автоматизації процесів бронювання в реальних бізнес-сценаріях. Запропоноване рішення може бути адаптоване для різних сфер надання послуг та використане як основа для подальшого розвитку функціональності.

Основні результати кваліфікаційної роботи були апробовані та оприлюднені у вигляді тез доповіді на X Міжнародній науково-практичній конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)», яка відбулася 23-24 травня 2025 року у місті Луцьк на базі Луцького національного технічного університету. Матеріали доповіді опубліковано у збірнику тез конференції: Тези доповідей X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)», Луцьк: ЛНТУ, 2025. С. 407-410 [1].

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ЗА ТЕМОЮ РОБОТИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

Сучасний розвиток технологій та потреби бізнесу ставлять на перший план автоматизацію рутинних процесів і підвищення ефективності взаємодії з кінцевими користувачами. Одним із найбільш ефективних інструментів для досягнення цих цілей є чат-боти, які здатні автоматизувати взаємодії, зменшити час на обробку запитів і значно покращити якість обслуговування клієнтів. Враховуючи величезний обсяг даних, що генеруються в різних сферах – від обслуговування клієнтів до управління послугами – чат-боти можуть стати важливим елементом інфраструктури бізнесу, замінюючи або значно полегшуючи роль людини у виконанні стандартних операцій. Чат-боти забезпечують не лише автоматизацію взаємодії з користувачем, а й надають можливості для більш гнучкої та персоналізованої роботи з інформацією, що, в свою чергу, підвищує рівень задоволення користувачів.

Незважаючи на значні досягнення в галузі розробки чат-ботів, проблеми, пов'язані з масштабованістю, інтеграцією з іншими інформаційними системами, а також необхідність збереження високого рівня безпеки та конфіденційності даних, залишаються актуальними. Зокрема, в умовах зростаючого обсягу транзакцій, таких як бронювання послуг або управління замовленнями, стає критично важливою правильна організація архітектури бота, його здатність масштабуватися в умовах високого навантаження та ефективно працювати з базами даних і зовнішніми сервісами. У цьому контексті інтеграція чат-ботів з сучасними хмарними платформами, такими як AWS, дозволяє вирішити проблеми масштабування і надійності, забезпечуючи безперебійну роботу в реальному часі навіть при великих обсягах даних і запитів.

Розробка чат-ботів – зокрема для месенджера Telegram – активно досліджується в контексті інформаційних систем, інтерфейсів взаємодії, обробки природної мови й автоматизації сервісів. Важливо розрізняти типи чат-ботів, їх архітектурні рішення, цільове призначення, а також підходи до оцінки якості. Нижче – огляд ключових досліджень, які дають системну картину сучасного стану галузі.

Одним із найбільш значущих напрямків у розвитку чат-ботів є застосування технологій штучного інтелекту (ШІ) для забезпечення адаптивної поведінки системи. Чат-боти, інтегровані з алгоритмами машинного навчання, дозволяють реалізувати гнучкі механізми взаємодії, які вивчають переваги та стиль спілкування користувачів.

У класичному оглядовому дослідженні [2] проаналізовано якісні та функціональні атрибути, які повинні характеризувати «добрі» чат-боти. Автори розглядають історію чат-ботів – від перших систем на кшталт ELIZA до сучасних «розумних агентів», – і підкреслюють, що завдяки зростанню відкритих кодів, платформ та доступу до NLP/ML технологій, створення ботів стало значно простішим.

У цій роботі запропоновано метод оцінки якості бота через набір критеріїв (зрозумілість відповіді, коректність, швидкість, адаптивність, юзабіліті тощо) із подальшим аналізом за допомогою методу AHP (Analytic Hierarchy Process). Це дослідження є фундаментальним для тих, хто хоче створити надійного, стабільного, «user-friendly» чат-бота – з точки зору як архітектури, так і UX/якості взаємодії. У контексті нашого проєкту (Telegram-бот для бронювання) такі критерії як надійність, швидкість відповіді, простота інтерфейсу, коректність логіки будуть критичними при оцінці успішності системи.

Чат-боти стали важливим інструментом у сфері обслуговування клієнтів та бронювання послуг. Вони дозволяють автоматизувати рутинні процеси, знижуючи навантаження на персонал і підвищуючи зручність для кінцевого користувача.

У дослідженні [3] описано реалізацію чат-бота, що дозволяє користувачам отримувати доступ до відкритих урядових/публічних даних через діалоговий інтерфейс. Бот підтримує багатомовність та дає змогу навіть не-експертам у сфері даних ефективно шукати й досліджувати інформацію.

Це дослідження демонструє, що чат-боти можуть виконувати не лише прості інформаційні функції, але й виступати як складні системи для навігації по великих базах даних – із логікою запит/відповідь, контекстом й фільтрацією. З точки зору нашої теми – це важливий досвід: бронювання послуг теж передбачає зберігання/обробку даних, вибір часу/дат, перевірку наявності, підтвердження й оновлення – тобто логіку, близьку до «пошуково-сервісної».

Застосування чат-ботів для бронювання послуг дозволяє знизити час обробки запитів, підвищити ефективність і зменшити ризик людських помилок. Такі боти можуть бути інтегровані з платіжними системами, що дозволяє забезпечити повний цикл бронювання – від вибору послуги до здійснення платежу.

У статті [4] здійснено огляд сучасних фреймворків і платформ для створення чат-ботів. Автори аналізують критерії відбору – функціональність, простоту розгортання, метрики ефективності, можливості масштабування. Враховуючи множину доступних рішень, вони надають рекомендації щодо оптимального вибору залежно від завдань: від простих інформаційних ботів.

Це дослідження важливе, бо воно систематизує ключові технічні аспекти створення чат-ботів, що допомагає обґрунтувати вибір (наприклад, фреймворку, бази даних, хостингу) у власному проєкті – саме те, що потрібно при розробці Telegram-бота для бронювання з використанням фреймворку, наприклад, на Node.js, і, можливо, у поєднанні з хмарними сервісами для зберігання/масштабування.

Згідно з аналізом у роботі [5], чат-боти – як програмні агенти – можуть стати універсальним інтерфейсом між людиною й інформаційними системами, замінюючи традиційні веб- або мобільні додатки. Проте автори

також звертають увагу на ризики: в деяких випадках чат-боти можуть бути використані для поширення неправдивої інформації або маніпуляцій.

Також підкреслюється потреба стандартизованих методів оцінки якості чат-ботів – як з технічної, так і з UX-точки зору. Саме тому обирати рішення для чат-бота потрібно з урахуванням не лише «що він робить», але й «як він це робить», – наскільки зручно, чи зрозуміло, чи надійно.

Особливо широко чат-боти застосовуються у сферах, де важлива швидка реакція на запити користувачів, таких як бронювання послуг, доставка товарів, а також підтримка клієнтів. Завдяки своєму простому інтерфейсу та можливості автоматизувати рутинні завдання, чат-боти допомагають знизити навантаження на персонал і забезпечити безперервну та ефективну взаємодію з клієнтами. Вивчення реальних прикладів використання чат-ботів у таких сферах, як бронювання авіаквитків, готелів, доставка їжі та підтримка клієнтів, дозволяє оцінити їх потенціал та можливості для оптимізації бізнес-процесів.

Приклади чат-ботів у сфері бронювання та послуг:

– чат-бот для бронювання авіаквитків Skyscanner [6], один з найбільших онлайн-сервісів для порівняння варіантів авіаперельотів, реалізував Telegram-бота, який дозволяє користувачам шукати та бронювати авіаквитки безпосередньо через чат. Бот запитує у користувача вихідні та кінцеві пункти маршруту, дату подорожі та кількість пасажирів. На основі цієї інформації, бот миттєво надає кілька варіантів рейсів, порівнюючи ціни та пропонуючи найбільш вигідні варіанти. Цей чат-бот демонструє зручність та ефективність використання Telegram як платформи для автоматизації бронювання. Завдяки інтеграції з базами даних авіакомпаній і наявності актуальних даних про рейси, користувачі можуть швидко знайти відповідний переліт без необхідності переходити на вебсайт. Це також дозволяє значно зменшити витрати часу на пошук і порівняння варіантів, що забезпечує високий рівень зручності для кінцевого користувача. Приклад із Skyscanner можна використовувати для аналізу функціональності чат-бота, який

інтегрується з зовнішніми базами даних і дозволяє автоматизувати вибір і бронювання послуг, що є важливим аспектом при розробці подібних ботів для інших видів послуг, таких як готелі або транспортні засоби;

– чат-бот для бронювання готелів Booking [7], провідний онлайн-сервіс для пошуку та бронювання готелів, також запустив чат-бота на платформі Telegram, який дозволяє користувачам здійснювати пошук і бронювання номерів у готелях. Бот спрощує процес бронювання, дозволяючи користувачеві вводити запит на пошук готелів у певному місті чи регіоні, а також вибирати дату заїзду та виїзду. Після цього бот надає список доступних готелів, з детальним описом кожного з них, наявністю вільних місць і цінами. Основною перевагою такого підходу є автоматизація процесу, що значно спрощує взаємодію користувача з платформою Booking.com. Користувачі не потребують відкривати сайт або мобільний додаток для бронювання – все необхідне доступне в рамках чат-бота, що робить процес більш швидким та зручним. Цей приклад можна розглядати як реальний кейс для дослідження того, як чат-боти можуть бути застосовані для автоматизації процесу бронювання готелів. Він також підкреслює важливість інтеграції чат-ботів з існуючими платформами і базами даних, що дозволяє користувачам отримувати актуальну інформацію в реальному часі.

– чат-бот для доставки їжі Uber Eats [8], популярний сервіс доставки їжі, реалізував чат-бота для здійснення замовлень через Telegram. Користувачі можуть вибрати ресторан і переглянути меню, а також здійснити замовлення прямо через чат. Бот надає деталі про доступні страви, пропонує кілька варіантів доставки і дає можливість оплатити замовлення безпосередньо через платформу. Цей чат-бот демонструє високий рівень інтерактивності та персоналізації в обслуговуванні клієнтів. Користувач може отримувати рекомендації на основі своїх попередніх замовлень, що дозволяє значно полегшити процес вибору страви. Крім того, чат-бот підтримує інтеграцію з платіжними системами, що робить процес доставки їжі ще більш зручним для кінцевого користувача. Цей приклад показує, як чат-боти можуть бути

використані в бізнесах, орієнтованих на доставку товарів і послуг. Важливою перевагою є інтеграція з системами оплати та реального часу, що дозволяє забезпечити безперебійний процес від вибору до доставки;

– чат-бот для підтримки клієнтів H&M, міжнародний рітейлер одягу, запустив чат-бота, що дозволяє надавати клієнтську підтримку через Telegram [9]. Бот допомагає вирішити питання, пов'язані з обміну товарів, поверненням товару, розмірами та наявністю продукції. Користувачі можуть звернутися до бота за відповідями на поширені запитання, отримати інструкції щодо процесу повернення товару або навіть отримати персоналізовану консультацію. Цей приклад показує, як чат-боти можуть не лише автоматизувати процеси бронювання, а й виконувати функції підтримки клієнтів, що є важливим аспектом для покращення взаємодії з користувачами. Чат-боти дозволяють значно знизити навантаження на операторів підтримки, одночасно підвищуючи ефективність і швидкість обробки запитів. Приклад із H&M є важливим для розуміння того, як чат-боти можуть виконувати роль консультантів для клієнтів, що актуально для широкого спектра сервісів, від магазинів до онлайн-ресурсів. Висновки Чат-боти для бронювання послуг та обслуговування клієнтів є потужним інструментом автоматизації бізнес-процесів. Вони дозволяють значно підвищити ефективність взаємодії між користувачами та сервісами, зменшити час на обробку запитів і значно покращити якість обслуговування. Реалізація чат-ботів у таких сферах, як бронювання авіаквитків, готелів, доставка їжі чи підтримка клієнтів, демонструє великий потенціал для розвитку подібних технологій. Важливим аспектом є інтеграція чат-ботів з зовнішніми базами даних, платіжними системами та іншими інструментами, що дозволяє створювати зручні й функціональні платформи для кінцевих користувачів.

Хмарні технології стали фундаментом для розвитку сучасних вебсервісів, включно з системами онлайн-бронювання. Їхнє впровадження дозволяє створювати розподілені, масштабовані та відмовостійкі рішення без необхідності утримання власної інфраструктури. Замість фізичних серверів

розробники використовують потужності хмарних провайдерів, таких як Amazon Web Services, Google Cloud Platform або Microsoft Azure, що надають готові сервіси для зберігання даних, обробки запитів та моніторингу.

У сфері бронювання хмарні технології мають особливе значення, адже такі системи зазвичай працюють у режимі реального часу, обробляючи запити від великої кількості користувачів. Кожна операція – від створення запису до надсилання повідомлення – повинна виконуватись швидко та без збоїв. Використання хмарної архітектури дозволяє розподіляти навантаження між серверами, автоматично масштабуватись у пікові години та зберігати дані в безпечному середовищі.

У сучасних рішеннях бронювання часто застосовується серверлес-архітектура – коли обчислення виконуються не на постійному сервері, а у вигляді незалежних функцій. Такий підхід дозволяє будувати високопродуктивні сервіси, які реагують лише тоді, коли надходить запит від користувача. Прикладом може слугувати зв'язка AWS Lambda та API Gateway, де перша відповідає за виконання логіки бронювання, а друга – за маршрутизацію запитів.

Amazon Web Services (AWS) – один із лідерів у галузі хмарних технологій, що пропонує широкий набір інструментів для побудови гнучких і безпечних систем.

У контексті систем бронювання найбільш затребуваними сервісами є:

– AWS Lambda – безсерверне виконання коду, що дозволяє масштабувати систему автоматично;

– Amazon API Gateway – керування запитами між клієнтами та сервером;

– Amazon RDS (PostgreSQL) – реляційна база даних для зберігання інформації про користувачів і бронювання [10];

– Amazon S3 – надійне сховище файлів (зображення, документи, квитанції);

– AWS SES або SNS – сервіси для розсилки сповіщень.

Використання цих сервісів дає змогу реалізувати модульну архітектуру, де кожен компонент (автентифікація, зберігання, обробка подій) функціонує незалежно, що полегшує оновлення та підтримку системи [11].

Боти Telegram можуть:

– взаємодіяти з користувачем через кнопки та меню;

– отримувати і відправляти файли, зображення, локації;

– реалізовувати багатокрокові сценарії (наприклад, послідовне бронювання);

– працювати з Telegram Payments API для обробки оплат.

Завдяки цьому Telegram стає ефективною платформою для впровадження сервісів бронювання, замовлень, консультацій та підтримки клієнтів [10].

На основі аналізу цих та інших джерел можна зробити кілька важливих висновків:

- чат-бот має бути спроектований з урахуванням якості взаємодії з користувачем (usability), коректності логіки, надійності та масштабованості – критерії, окреслені в [2];

- якщо бот передбачає обробку даних, зберігання записів, запитів – доцільно звернутися до досвіду, як у [3], що демонструє, як чат-бот може бути «проводирем» до бази даних або зовнішнього сервісу;

- вибір технологій та фреймворку має базуватися на порівняльному аналізі платформ і метрик, як у [4] – це дає змогу обґрунтувати, чому саме обрано, наприклад, Node.js/NestJS, певну БД чи хостинг;

- водночас слід враховувати обмеження чат-ботів: вони підходять не для всіх типів інтерфейсів або складних UI-задач; для деяких сценаріїв може бути доцільнішим веб-інтерфейс чи мобільний додаток.

1.2 Огляд і аналіз методів та засобів розробки Telegram-бота для бронювання послуг із використанням NestJS та AWS сервісів для вирішення проблеми дослідження

Під час розробки систем, що взаємодіють із користувачами через месенджери, важливо забезпечити не лише зручність спілкування, а й надійну серверну логіку, здатну обробляти велику кількість запитів. Сучасні архітектурні підходи до побудови таких рішень охоплюють:

– монолітну архітектуру, коли весь застосунок розміщений у межах одного сервера. Перевагою є простота розробки, але недоліком – складність масштабування;

– мікросервісну архітектуру, у якій система поділяється на незалежні модулі (автентифікація, бронювання, аналітика). Це полегшує підтримку, дозволяє масштабувати окремі частини, але вимагає чіткої взаємодії між ними;

– серверлес-архітектуру, коли кожен виклик функції виконується окремо у хмарі (наприклад, AWS Lambda). Цей підхід забезпечує гнучке масштабування і зменшення витрат на інфраструктуру, оскільки ресурси оплачуються лише під час виконання запитів.

Для Telegram-бота, що обробляє події через Webhook, оптимальним рішенням є серверлес-архітектура, оскільки вона дозволяє реагувати на запити користувачів у реальному часі без постійного утримання серверів.

Під час розробки системи бронювання за допомогою Telegram-бота важливо обрати оптимальний набір технологій, які забезпечать баланс між простотою реалізації, гнучкістю архітектури та стабільністю в роботі. Вибір технологічного стеку грунтувався на трьох ключових принципах: масштабованість, ефективність розробки та інтеграція з хмарними сервісами.

NestJS став центральним елементом серверної частини проєкту завдяки своїй модульній структурі, що базується на концепціях dependency injection і декораторів. Цей фреймворк дозволяє розділити логіку застосунку на

незалежні модулі, що значно спрощує розробку та тестування. На відміну від Express.js, який вимагає ручного структурування, NestJS пропонує сувору архітектуру, яка відповідає принципам розробки корпоративних систем [11].

Мова програмування TypeScript відіграє не менш важливу роль, оскільки забезпечує статичну типізацію та автодоповнення, що зменшує кількість помилок на етапі компіляції. Завдяки цьому можливо ефективно розробляти великі проєкти з чітко визначеними інтерфейсами між компонентами.

У якості бази даних обрано PostgreSQL, розгорнуту у сервісі Amazon RDS. Це рішення забезпечує надійність, автоматичне резервне копіювання, високу доступність і масштабування за потреби. Доступ до даних реалізовано через TypeORM, що дозволяє працювати з моделями об'єктно-реляційного типу [12].

Щодо хмарних сервісів, використання AWS Lambda і API Gateway дозволило реалізувати серверлес-підхід, за якого код виконується лише при наявності запиту. Це мінімізує витрати на інфраструктуру та спрощує розгортання. Крім того, AWS надає комплексну екосистему для моніторингу, авторизації, зберігання даних і безпеки.

Таким чином, комбінація NestJS + TypeScript + AWS забезпечує високу ефективність у розробці, прозору архітектуру та можливість гнучкої інтеграції з Telegram API. Такий вибір дозволяє створити не лише функціональний бот, але й готовий бекенд для масштабованих рішень бронювання.

На сьогодні існує велика кількість бібліотек і фреймворків, що спрощують створення Telegram-ботів у середовищі Node.js. Вибір конкретного інструменту залежить від складності проєкту, вимог до масштабованості, підтримки middleware та інтеграції з іншими компонентами системи.

Одним із найпоширеніших варіантів є node-telegram-bot-api – стабільна бібліотека з великим ком'юніті, що забезпечує базову взаємодію з Telegram API. Вона добре підходить для невеликих ботів, але має обмеження у побудові

складної логіки, оскільки не підтримує структурування коду за принципами сучасних фреймворків.

Більш гнучким рішенням є Telegraf.js – фреймворк, який підтримує асинхронну обробку запитів, систему middleware та контекстні сцени. Завдяки цьому розробники можуть створювати ботів із багаторівневою логікою, розгалуженими діалогами та інтеграціями з REST API. Саме Telegraf став стандартом де-факто для Node.js спільноти.

Ще одним підходом є використання NestJS Telegram Module, який забезпечує інтеграцію Telegram API безпосередньо у модульну архітектуру NestJS. Це дозволяє скористатися всіма перевагами dependency injection, декораторів та сервісів NestJS, але через відносну молодість модуля він має менше готових прикладів і документації [11].

У межах даного проєкту було обрано Telegraf.js як базову бібліотеку для роботи з Telegram API, інтегровану у середовище NestJS через адаптер. Такий підхід поєднує простоту і гнучкість Telegraf із впорядкованістю архітектури NestJS, забезпечуючи:

- модульність коду;
- повторне використання сервісів та middleware;
- централізовану обробку помилок;
- легку інтеграцію з AWS Lambda.

Це рішення дає змогу розробляти масштабовані Telegram-боти, які залишаються читабельними і підтримуваними навіть у великих проєктах.

Telegram API працює за моделлю Webhook, тобто сервер (або Lambda-функція) отримує повідомлення напряму з Telegram-серверів при кожній дії користувача [13].

AWS API Gateway приймає цей запит і викликає Lambda-функцію, яка:

- аналізує тип повідомлення (команда, кнопка, текст);
- обробляє дані через бізнес-логіку (наприклад, створює бронювання);
- записує інформацію в базу Amazon RDS;
- повертає відповідь користувачу через Telegram API.

Така схема дозволяє системі залишатись повністю безсерверною і масштабуватись автоматично залежно від кількості користувачів.

Хмарна інфраструктура Amazon Web Services відіграє ключову роль у забезпеченні стабільності та масштабованості системи бронювання. У даному проєкті було використано кілька основних сервісів AWS, кожен з яких виконує окрему функцію в межах архітектури.

Центральним елементом є AWS Lambda, що дозволяє запускати код без постійного утримання серверів. Кожна Lambda-функція виконує конкретну задачу: обробку команд користувача, запис бронювання, надсилання сповіщень або взаємодію з базою даних. Завдяки цьому система може гнучко масштабуватись залежно від потоку запитів, а її робота оплачується лише за фактичний час виконання.

Amazon API Gateway виступає проміжною ланкою між Telegram і Lambda. Він приймає HTTPS-запити, що надходять через Webhook, і маршрутизує їх до відповідних функцій. Крім того, API Gateway забезпечує базову авторизацію, кешування відповідей і моніторинг трафіку.

База даних реалізована у Amazon RDS на основі PostgreSQL, що дозволяє автоматично виконувати резервне копіювання, оновлення та масштабування. Для зберігання файлів і медіа використовується Amazon S3, який є стійким і безпечним сховищем даних. Важливим компонентом є також AWS CloudWatch, який надає розробнику змогу відстежувати роботу кожної функції, фіксувати помилки та аналізувати продуктивність системи [10].

Інтеграція всіх цих сервісів забезпечує єдиний життєвий цикл запиту – від моменту, коли користувач вводить команду в Telegram, до запису бронювання у базі даних і зворотного повідомлення. Такий підхід дозволяє зосередитися на логіці застосунку, мінімізуючи адміністративні завдання, пов'язані з утриманням серверів.

Архітектура системи Telegram-бота для бронювання побудована на принципах модульності, масштабованості та мінімального зв'язування компонентів. Основна мета полягає у тому, щоб забезпечити ефективну

взаємодію між користувачем у Telegram, серверною частиною та хмарними сервісами AWS.

Загалом система складається з кількох логічних рівнів:

– рівень представлення (Presentation Layer) – Telegram-бот, через якого користувач взаємодіє із системою. Бот приймає команди, повідомлення та кнопкові події, які потім передаються до серверної логіки. Для реалізації цього рівня використовується бібліотека Telegraf.js, інтегрована у NestJS;

– рівень бізнес-логіки (Application Layer) – модулі, які обробляють запити користувача, перевіряють доступність послуг, створюють записи бронювання і взаємодіють із базою даних. Цей шар реалізований за допомогою NestJS як набору сервісів і контролерів, які працюють у середовищі AWS Lambda;

– рівень даних (Data Layer) – включає Amazon RDS (PostgreSQL) для зберігання структурованої інформації (користувачі, послуги, розклади) та Amazon S3 для зберігання статичних ресурсів, наприклад, зображень чи документів підтвердження бронювання;

– рівень моніторингу та логування (Monitoring Layer) – представлений сервісом AWS CloudWatch, який забезпечує збір логів, аналітику виконання Lambda-функцій і моніторинг продуктивності системи.

На рисунку 1.1 зображено загальну архітектуру системи. Схема демонструє основні компоненти та логіку взаємодії між ними.

Взаємодія між цими рівнями відбувається наступним чином: коли користувач надсилає команду в Telegram, API Telegram надсилає HTTP-запит на адресу AWS API Gateway, який маршрутизує його до відповідної Lambda-функції. Lambda обробляє дані, виконує бізнес-логіку, звертається до бази даних і повертає відповідь користувачу через Telegram API. Такий підхід дозволяє мінімізувати затримки, зберігаючи при цьому незалежність кожного компоненту.

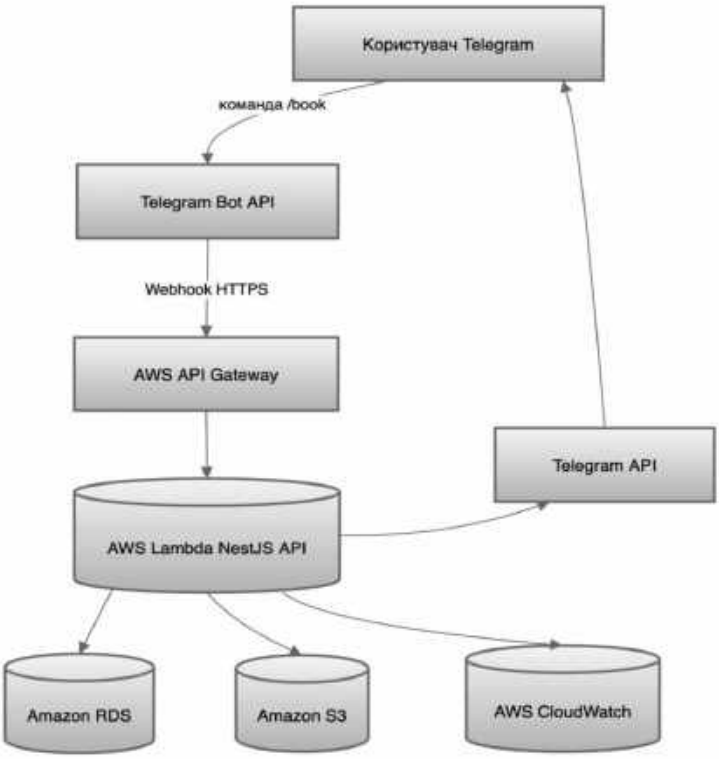

Рисунок 1.1 – Діаграма взаємодії сервісів

Основні переваги запропонованої архітектури:

– масштабованість – AWS автоматично виділяє додаткові ресурси у разі підвищеного навантаження;

– надійність – навіть у випадку відмови однієї функції інші продовжують працювати незалежно;

– простота обслуговування: кожен компонент можна оновлювати або розгортати окремо;

– економічність – оплата здійснюється лише за фактичний час виконання запитів, що робить проєкт рентабельним.

Крім того, така архітектура легко адаптується до нових вимог. Наприклад, у майбутньому можна додати підтримку інших каналів взаємодії – вебінтерфейсу, мобільного застосунку або інтеграції з календарями Google чи Outlook. Завдяки модульній побудові NestJS і гнучкості AWS, система залишатиметься стійкою до змін без необхідності повного переписування коду.

Таким чином, архітектура застосунку поєднує сучасні підходи до розробки серверлес-систем, що забезпечує ефективну, безпечну і масштабовану інфраструктуру для надання онлайн-послуг у реальному часі.

Хмарна інфраструктура Amazon Web Services відіграє ключову роль у забезпеченні стабільності та масштабованості системи бронювання. У даному проєкті було використано кілька основних сервісів AWS, кожен з яких виконує окрему функцію в межах архітектури.Центральним елементом є AWS Lambda, що дозволяє запускати код без постійного утримання серверів. Кожна Lambda-функція виконує конкретну задачу: обробку команд користувача, запис бронювання, надсилання сповіщень або взаємодію з базою даних. Завдяки цьому система може гнучко масштабуватись залежно від потоку запитів, а її робота оплачується лише за фактичний час виконання.

Amazon API Gateway виступає проміжною ланкою між Telegram і Lambda. Він приймає HTTPS-запити, що надходять через Webhook, і маршрутизує їх до відповідних функцій. Крім того, API Gateway забезпечує базову авторизацію, кешування відповідей і моніторинг трафіку [10].

База даних реалізована у Amazon RDS на основі PostgreSQL, що дозволяє автоматично виконувати резервне копіювання, оновлення та масштабування. Для зберігання файлів і медіа використовується Amazon S3, який є стійким і безпечним сховищем даних. Важливим компонентом є також AWS CloudWatch, який надає розробнику змогу відстежувати роботу кожної функції, фіксувати помилки та аналізувати продуктивність системи.

Інтеграція всіх цих сервісів забезпечує єдиний життєвий цикл запиту – від моменту, коли користувач вводить команду в Telegram, до запису бронювання у базі даних і зворотного повідомлення. Такий підхід дозволяє зосередитися на логіці застосунку, мінімізуючи адміністративні завдання, пов'язані з утриманням серверів.

Таким чином, для реалізації Telegram-бота системи бронювання було обрано сучасний стек технологій, що забезпечує продуктивність, безпеку та гнучкість архітектури.

Використання AWS у поєднанні з NestJS дозволяє будувати «серверлес» рішення корпоративного рівня, з мінімальними витратами на підтримку. Цей підхід відповідає актуальним тенденціям індустрії розробки – переходу до хмарних, подієвих і безсерверних архітектур.

1.3 Постановка завдання на кваліфікаційну роботу магістра

Для досягнення мети кваліфікаційної роботи та комплексного розв'язання поставленої науково-практичної проблеми було сформульовано низку завдань, виконання яких забезпечує поетапну реалізацію дослідження – від аналізу теоретичних і практичних підходів до розробки та експериментальної перевірки програмного рішення. Зазначені завдання визначають логіку структури роботи та слугують основою для обґрунтування прийнятих технологічних і архітектурних рішень:

– проаналізувати сучасні наукові дослідження та існуючі програмні рішення у сфері автоматизації бронювання послуг із використанням чат-ботів та хмарних технологій;

– обґрунтувати вибір технологій, фреймворків і хмарних сервісів для реалізації Telegram-боту бронювання послуг;

– спроєктувати структуру даних та розробити модель реляційної бази даних для зберігання інформації про бізнеси, послуги, часові інтервали та бронювання;

– розробити Telegram-бот для бронювання послуг із реалізацією багатоетапного сценарію взаємодії користувача та підтримкою нагадувань. Інтегрувати розроблений Telegram-бот у безсерверну хмарну інфраструктуру Amazon Web Services;

– провести експериментальне тестування розробленого Telegram-боту та проаналізувати результати його функціонування.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ TELEGRAM-БОТУ ДЛЯ БРОНЮВАННЯ ПОСЛУГ ІЗ ВИКОРИСТАННЯМ NESTJS ТА AWS СЕРВІСІВ

2.1 Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

Для реалізації Telegram-боту бронювання послуг було обрано сучасний технологічний стек, орієнтований на масштабованість, багатокористувацьку взаємодію та ефективну роботу в хмарному безсерверному середовищі. Основним серверним фреймворком виступає NestJS, який забезпечує модульну архітектуру, чітке розмежування відповідальностей між компонентами системи та підтримку ін'єкції залежностей. Використання NestJS дозволяє структурувати бізнес-логіку у вигляді сервісів і контролерів, що спрощує супровід коду та його подальше розширення [14].

Для взаємодії з платформою Telegram застосовано бібліотеку Telegraf, яка надає зручний програмний інтерфейс для обробки команд, повідомлень, callback-запитів і веб-хуків. Обґрунтованим є вибір саме цього інструменту, оскільки він добре інтегрується з серверними застосунками на Node.js та дозволяє динамічно керувати кількома екземплярами ботів. Це дало змогу реалізувати багатокористувацьку (multi-tenant) архітектуру, за якої різні бізнеси можуть використовувати одну спільну кодову базу, але з ізольованими логічними екземплярами Telegram-ботів, що ідентифікуються за унікальними токенами.

З огляду на вимоги до масштабованості та економії ресурсів було обрано безсерверну архітектуру на базі сервісів Amazon Web Services. Зокрема, виконання серверної частини здійснюється за допомогою AWS Lambda, а прийом запитів від Telegram організовано через Amazon API Gateway. Для адаптації NestJS-застосунку до безсерверного середовища використано підхід інтеграції Express-додатку з AWS Lambda, що дозволяє

поєднати повнофункціональний серверний фреймворк із перевагами serverless-моделі. Додатково застосовано кешування обробника Lambda-функції з метою зменшення затримок під час повторних викликів.

Однією з ключових задач при розробці чат-ботів у безсерверному середовищі є підтримка стану діалогу, оскільки такі середовища не зберігають стан між запитами. Для розв'язання цієї проблеми використано алгоритм керування станами у вигляді скінченного автомата, реалізованого через зберігання поточного стану бронювання користувача в оперативній пам'яті сервісу. Структура стану містить ідентифікатори бізнесу, послуги та таймслоту, а також інформацію про поточний крок діалогу, коментар користувача та параметри нагадування. Такий підхід забезпечує послідовність взаємодії з користувачем та дозволяє коректно обробляти багатоетапний процес бронювання.

Для зберігання постійних даних використано реляційну базу даних з доступом через ORM-бібліотеку TypeORM. Це рішення дозволяє описувати структуру бази даних у вигляді класів-сутностей, визначати зв'язки між ними та забезпечувати цілісність даних [15]. У системі реалізовано сутності для бізнесів, послуг, бронювань і таймслотів, а також використано перелічувані типи для представлення статусів бронювання. Такий підхід спрощує роботу з даними та зменшує ризик логічних помилок при виконанні операцій створення або скасування бронювання.

Бізнес-логіка бронювання реалізована з урахуванням узгодженості даних, зокрема шляхом автоматичного позначення відповідних часових інтервалів як недоступних після підтвердження бронювання та їх повторного відкриття у разі скасування. Це забезпечує коректне управління ресурсами та запобігає дублюванню бронювань на один і той самий таймслот. Завершальним етапом кожної сесії бронювання є очищення тимчасового стану користувача, що запобігає накопиченню неактуальних даних у системі.

Таким чином, обрані шляхи, технології та алгоритми дозволили створити масштабований Telegram-бот для бронювання послуг, який поєднує

безсерверну архітектуру AWS, модульний підхід NestJS та ефективні механізми управління станом діалогу й даними, що відповідає сучасним вимогам до хмарних інформаційних систем.

На рисунку 2.1 представлено UML-діаграму послідовностей, яка відображає процес взаємодії користувача з Telegram-ботом для бронювання послуг, розробленим із використанням фреймворку NestJS та хмарних сервісів AWS. Діаграма демонструє послідовність обміну повідомленнями між основними компонентами системи та ілюструє логіку обробки користувацьких запитів на різних етапах формування бронювання.

Рисунок 2.1–Діаграма послідовності «Як підтримується стан діалогу між повідомленнями»

Процес розпочинається з ініціації взаємодії користувачем шляхом надсилання команди /start, після чого Telegram-бот обробляє запит і ініціює відповідний метод сервісного рівня. Користувачу відображається перелік доступних послуг, з яких він обирає необхідну. Після вибору послуги сервісний компонент зберігає поточний стан бронювання у спеціальній структурі даних bookingStates Map, що виконує роль тимчасового сховища та забезпечує керування станами діалогу відповідно до концепції скінченного автомата.

На наступному етапі користувачу пропонується ввести додатковий коментар до бронювання. Введені дані обробляються ботом та зберігаються у стані користувача, після чого система переходить до етапу налаштування нагадування. Користувач має можливість обрати, чи бажає він отримувати нагадування про заплановану послугу. У разі позитивної відповіді система переходить до етапу вибору часу нагадування, де користувач визначає кількість хвилин до початку послуги, за яку має бути надіслане повідомлення.

Після завершення введення всіх необхідних параметрів бот формує зведення бронювання та відображає його користувачу для підтвердження. Підтвердження ініціює виклик сервісу створення бронювання, який на основі збереженого стану формує відповідний об'єкт бронювання та передає його до компонента роботи з базою даних. На цьому етапі відбувається збереження інформації про бронювання та позначення відповідного часового інтервалу як недоступного.

Після успішного створення бронювання тимчасовий стан користувача видаляється зі сховища bookingStates Map, що запобігає накопиченню неактуальних даних. Завершальним кроком є надсилання користувачу повідомлення про успішне підтвердження бронювання. Таким чином, діаграма відображає повний життєвий цикл процесу бронювання – від ініціації взаємодії до збереження даних у базі та очищення проміжного стану,

що забезпечує узгоджену та масштабовану роботу Telegram-бота в хмарному середовищі.

2.2 Практична реалізація об'єкта проектування

Процес розробки Telegram-боту для бронювання послуг було організовано поетапно з урахуванням функціональних вимог системи, особливостей безсерверної архітектури та специфіки взаємодії користувачів у чат-інтерфейсі. На початковому етапі виконано аналіз предметної області, в результаті якого визначено основні сутності системи: бізнеси, послуги, часові інтервали, користувачі та бронювання. На основі цього аналізу сформовано логічну модель даних, яка стала основою для подальшого проєктування бази даних.

На етапі архітектурного проєктування було обрано багатокористувацьку (multi-tenant) модель, що дозволяє кільком незалежним бізнесам використовувати одну спільну кодову базу Telegram-боту. Для цього реалізовано механізм динамічної ініціалізації ботів на основі унікальних Telegram Bot Token. Приклад відповідної реалізації наведено в лістингу 2.1, де показано створення та кешування екземплярів ботів для конкретних бізнесів.

Лістинг 2.1 – Ініціалізація Telegram-бота для конкретного бізнесу

```
// Initialize bot for a specific business token
async initializeBot(business: Business): Promise<void> {
 if (this.bots.has(business.id)) {
  return; // Bot already initialized
 }

 const bot = new Telegraf(business.telegramBotToken);
 this.setupHandlers(bot, business);
 await this.setBotCommands(bot, business);

 this.bots.set(business.id, bot);
 this.businessCache.set(business.id, business);
 this.logger.log(`Bot initialized for business: ${business.name}`);
```

```
}

// Get or create bot instance for a business
private async getBotForBusiness(businessId: string): Promise<Telegraf> {
 if (this.bots.has(businessId)) {
  return this.bots.get(businessId)!;
 }

 const business = await this.businessService.getBusinessById(businessId);
 if (!business) {
  throw new Error(`Business ${businessId} not found`);
 }

 await this.initializeBot(business);
 return this.bots.get(businessId)!;
}
```

кінець лістингу 2.1

На рисунку 2.2 зображено Multi-tenant архітектуру, як кілька бізнесів використовують одну базу коду з ізольованими екземплярами ботів.

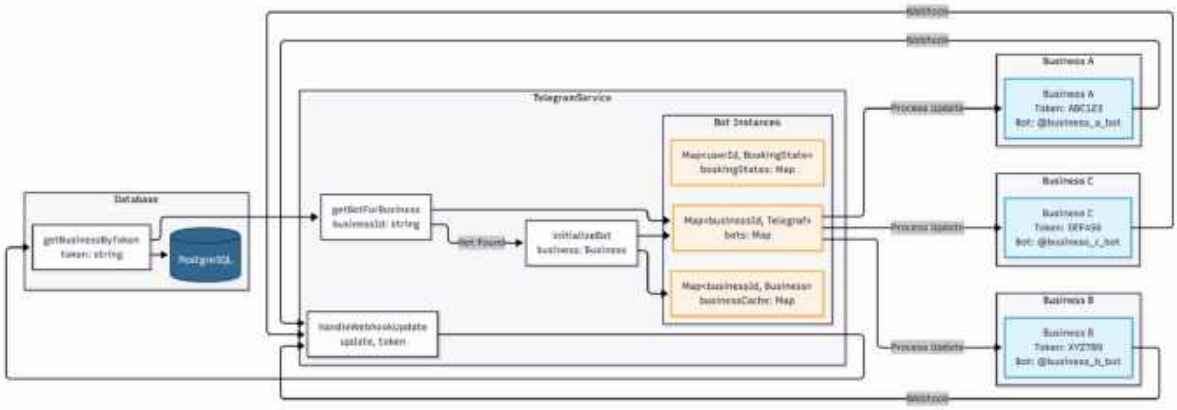

Рисунок 2.2 – Multi-tenant архітектура

Наступним етапом стала реалізація серверної логіки із використанням фреймворку NestJS. Архітектура застосунку побудована за модульним принципом із чітким розмежуванням відповідальностей між контролерами та сервісами. Для прийому оновлень від Telegram реалізовано контролер веб-хуків, який забезпечує маршрутизацію запитів до відповідного екземпляра

бота та перевірку секретного токена. Відповідний механізм продемонстровано в лістингу 2.2.

Лістинг 2.2 – Контролер обробки Telegram веб-хуків з маршрутизацією за токеном

```
@Post('webhook')
async handleWebhook(
 @Body() body: any,
 @Headers('x-telegram-bot-api-secret-token') secretToken?: string,
 @Query('token') botToken?: string,
) {
 // Verify webhook secret token if configured
 const expectedToken = process.env.WEBHOOK_SECRET_TOKEN;
 if (expectedToken && secretToken !== expectedToken) {
  this.logger.warn('Invalid webhook secret token');
  return { ok: false, error: 'Unauthorized' };
 }

 try {
  // Route to correct bot based on token
  await this.telegramService.handleWebhookUpdate(body, botToken);
  return { ok: true };
 } catch (error) {
  this.logger.error('Error processing webhook:', error);
  return { ok: false, error: error.message };
 }
}
```

кінець лістингу 2.2

Для забезпечення масштабованої та надійної роботи Telegram-боту було спроєктовано хмарну інфраструктуру на базі сервісів Amazon Web Services. Архітектурне рішення передбачає використання безсерверної моделі, у якій основна серверна логіка виконується за допомогою AWS Lambda, а прийом зовнішніх HTTP-запитів організовано через Amazon API Gateway. Такий підхід дозволяє автоматично масштабувати систему залежно від навантаження та зменшити витрати на утримання серверних ресурсів. Для забезпечення безпеки та контрольованого доступу до компонентів інфраструктури налаштовано мережеві параметри, включно з віртуальною приватною мережею, правилами доступу та групами безпеки. Загальна

конфігурація використаних AWS-сервісів, мережевих компонентів і механізмів захисту представлена на рисунку 2.3.

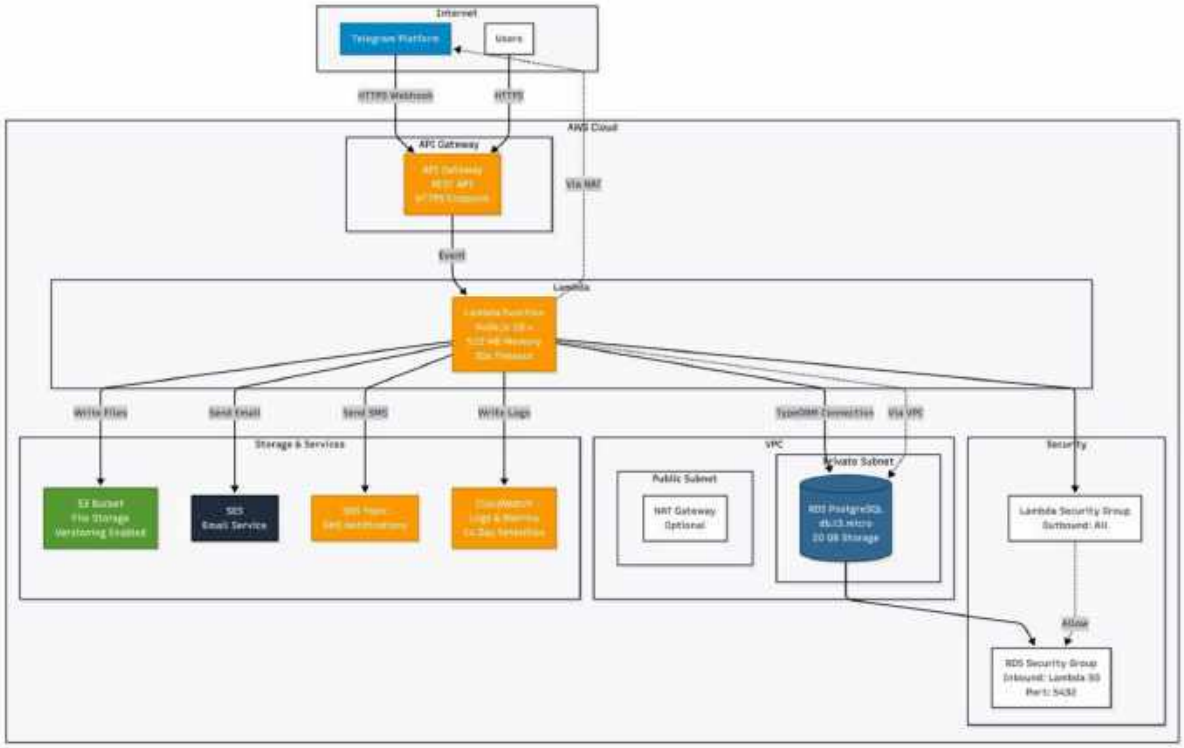

Рисунок 2.3 –Інфраструктура AWS. Конфігурація служб AWS, мереж та груп безпеки

Оскільки застосунок розгортається у безсерверному середовищі, де відсутнє збереження стану між запитами, однією з ключових задач стало управління станом діалогу користувача. Для її вирішення реалізовано алгоритм керування станами у вигляді скінченного автомата, де кожен крок процесу бронювання відповідає окремому логічному стану. Поточний стан користувача зберігається в оперативній пам'яті сервісу у структурі Map<userId, BookingState>. Реалізацію цього підходу наведено в лістингу 2.3.

Лістинг 2.3 – Реалізація керування станом діалогу користувача

```
interface BookingState {
  businessId: string;
  timeslotId: string;
  serviceId: string;
```

```
  step: BookingStep;
  comment?: string;
  wantsReminder?: boolean;
  reminderMinutesBefore?: number;
 }

 @Injectable()
 export class TelegramService {
  private bookingStates: Map<number, BookingState> = new Map(); // userId ->
BookingState

  // Usage in confirmBooking method
  private async confirmBooking(ctx: Context) {
   const userId = ctx.from?.id;
   const state = this.bookingStates.get(userId);
   if (!state) {
    // Handle error
    return;
   }

   // Use state for booking creation
   const booking = await this.bookingService.createBooking({
    userId,
    businessId: state.businessId,
    serviceId: state.serviceId,
    timeslotId: state.timeslotId,
    comment: state.comment,
    wantsReminder: state.wantsReminder || false,
    reminderMinutesBefore: state.reminderMinutesBefore || null,
   });

   // Clear state after completion
   this.bookingStates.delete(userId);
  }
 }
```

кінець лістингу 2.3

Паралельно з реалізацією логіки діалогу було спроєктовано та реалізовано рівень доступу до даних. Для цього використано ORM-бібліотеку TypeORM, за допомогою якої створено сутності бізнесів, послуг, таймслотів, користувачів та бронювань із відповідними зв'язками. Приклад визначення сутностей і зв'язків між ними наведено в лістингу 2.4.

Лістинг 2.4 – Опис сутностей бази даних та зв'язків між ними

```
// Business Entity
@Entity('businesses')
export class Business {
 @PrimaryGeneratedColumn('uuid')
 id: string;

 @Column({ unique: true, name: 'telegram_bot_token' })
 telegramBotToken: string;

 @Column({ type: 'varchar', length: 10, default: 'uk_UA', name: 'locale' })
 locale: string;

 @OneToMany(() => Service, (service) => service.business)
 services: Service[];

 @OneToMany(() => Booking, (booking) => booking.business)
 bookings: Booking[];
}

// Booking Entity with Status Enum
@Entity('bookings')
export class Booking {
 @ManyToOne(() => Business, (business) => business.bookings)
 @JoinColumn({ name: 'business_id' })
 business: Business;

 @Column({
  type: 'varchar',
  default: BookingStatus.PENDING,
  name: 'status',
 })
 status: BookingStatus;
}
```

кінець лістингу 2.4

Для забезпечення збереження та узгодженого управління даними в системі Telegram-боту для бронювання послуг було спроєктовано реляційну модель бази даних. Під час проєктування структури бази даних враховано вимоги до багатокористувацької архітектури, цілісності даних та підтримки основних бізнес-процесів системи, зокрема керування послугами, часовими інтервалами та бронюваннями. Модель даних базується на використанні взаємопов'язаних сутностей із чітко визначеними первинними та зовнішніми ключами, що забезпечує коректну реалізацію зв'язків між бізнесами,

користувачами та їх бронюваннями. Загальну структуру бази даних та взаємозв'язки між її основними таблицями наведено на рисунку 2.4.

Рисунок 2.4 – Схема бази даних

Бізнес-логіка бронювання реалізована з урахуванням узгодженості даних та управління ресурсами. Під час створення бронювання відповідний часовий інтервал автоматично позначається як недоступний, що запобігає виникненню конфліктів. Аналогічно, у разі скасування бронювання таймслот знову стає доступним. Відповідні алгоритми створення та скасування бронювання наведено в лістингах 2.5 та 2.6.

Лістинг 2.5 – Створення бронювання з оновленням доступності

```
async createBooking(dto: CreateBookingDto): Promise<Booking> {
 const booking = this.bookingRepository.create({
  userId: dto.userId,
```

```
    businessId: dto.businessId,
    serviceId: dto.serviceId,
    timeslotId: dto.timeslotId,
    comment: dto.comment,
    wantsReminder: dto.wantsReminder || false,
    reminderMinutesBefore: dto.reminderMinutesBefore || null,
    status: BookingStatus.CONFIRMED,
  });

  // Mark timeslot as unavailable
  const timeslot = await this.getTimeslotById(dto.timeslotId);
  if (timeslot) {
    timeslot.isAvailable = false;
    await this.timeslotRepository.save(timeslot);
  }

  return this.bookingRepository.save(booking);
}
```

кінець лістингу 2.5

Лістинг 2.6 – Скасування бронювання з вивільненням ресурсів

```
async cancelBooking(bookingId: string, userId: number): Promise<Booking> {
  const booking = await this.bookingRepository.findOne({
    where: { id: bookingId, userId },
    relations: ['timeslot'],
  });

  if (!booking) {
    throw new Error('Booking not found');
  }

  if (booking.status === BookingStatus.CANCELLED) {
    throw new Error('Booking is already cancelled');
  }

  if (booking.status === BookingStatus.COMPLETED) {
    throw new Error('Cannot cancel a completed booking');
  }
  // Update booking status
  booking.status = BookingStatus.CANCELLED;
  await this.bookingRepository.save(booking);

  // Mark timeslot as available again
  if (booking.timeslot) {
    booking.timeslot.isAvailable = true;
    await this.timeslotRepository.save(booking.timeslot);
  }
  return booking;
}
```

кінець лістингу 2.6

Завершальним етапом процесу розробки стала адаптація застосунку до безсерверної архітектури Amazon Web Services. Для цього реалізовано AWS Lambda Handler з інтеграцією NestJS та кешуванням обробника, що дозволяє зменшити затримки під час повторних викликів. Приклад відповідної реалізації наведено в лістингу 2.7.

Лістинг 2.7 – AWS Lambda Handler з інтеграцією NestJS

```
import { NestFactory } from '@nestjs/core';
import { ExpressAdapter } from '@nestjs/platform-express';
import express from 'express';
import serverlessExpress from '@vendia/serverless-express';

let cachedHandler: any;

async function bootstrapServer() {
 const expressApp = express();
 const adapter = new ExpressAdapter(expressApp);
 const app = await NestFactory.create(AppModule, adapter);
 app.useGlobalPipes(new ValidationPipe());
 app.enableCors();
 await app.init();
 return serverlessExpress({ app: expressApp });
}

export const handler = async (
 event: APIGatewayProxyEvent,
 context: Context,
): Promise<APIGatewayProxyResult> => {
 if (!cachedHandler) {
  cachedHandler = await bootstrapServer();
 }
 return cachedHandler(event, context);
};
```

кінець лістингу 2.7

Для наочної ілюстрації результатів практичної реалізації розробленого Telegram-боту наведено приклад взаємодії користувача з системою в реальному чаті. У представленому сценарії продемонстровано основні етапи процесу бронювання послуг, зокрема вибір сервісу, введення додаткової інформації, налаштування параметрів нагадування та підтвердження бронювання. Такий приклад дозволяє оцінити коректність реалізації бізнес-

логіки, послідовність діалогу та зручність користувацького інтерфейсу. Фрагмент чату, що відображає взаємодію користувача з Telegram-ботом у процесі бронювання, наведено на рисунках 2.5-2.7.

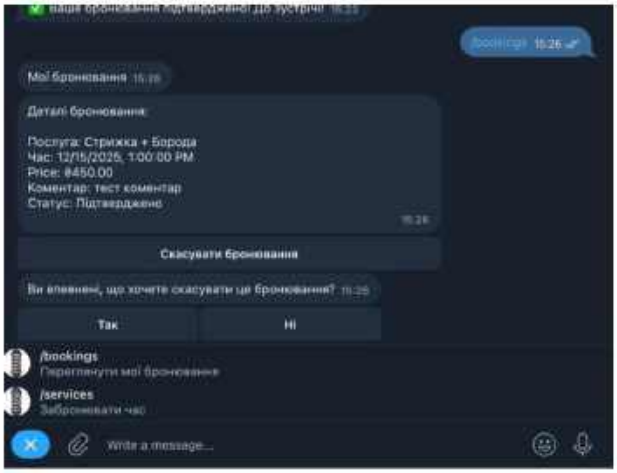

Рисунок 2.5 – Інтерфейс керування бронюваннями користувача

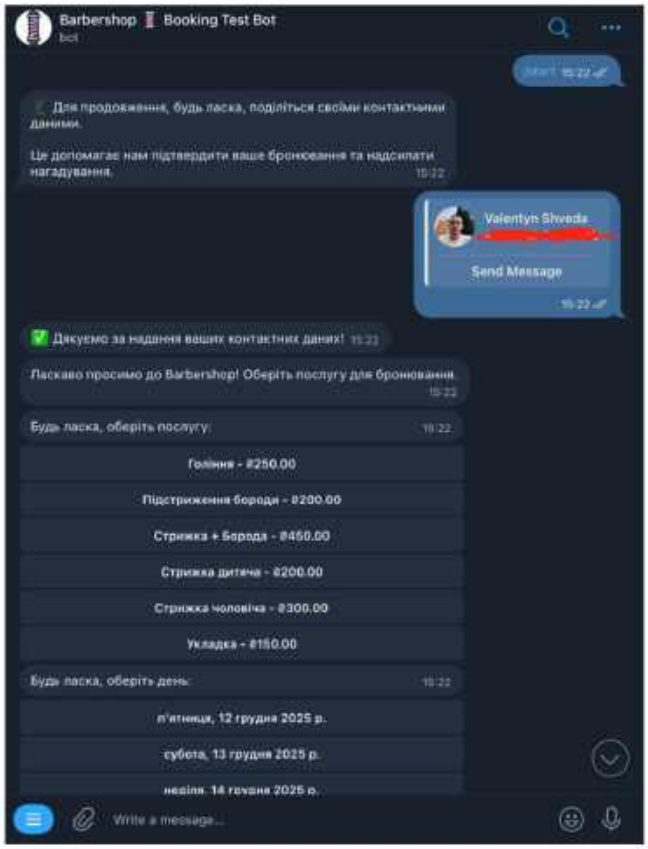

Рисунок 2.6 – Початковий етап взаємодії користувача з Telegram-ботом та вибір послуги

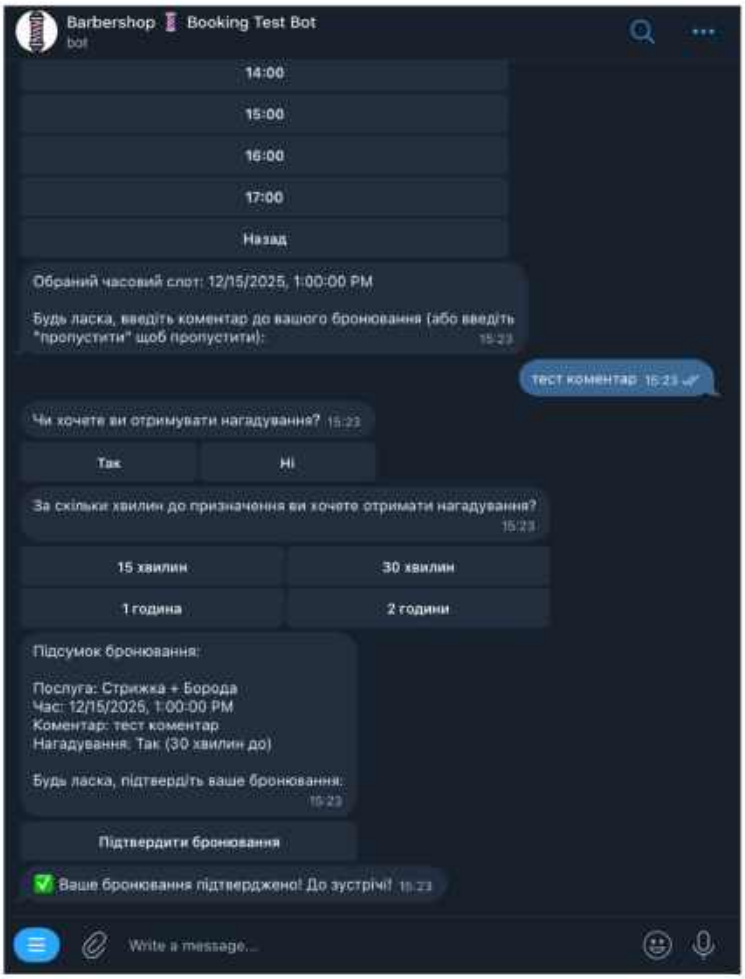

Рисунок 2.7 – Інтерфейс підтвердження бронювання та вибору параметрів нагадування

Таким чином, процес розробки об’єкта проєктування охоплював послідовні етапи аналізу, архітектурного проєктування, реалізації серверної логіки, управління станом діалогу, роботи з базою даних та інтеграції з хмарною інфраструктурою AWS. Це створює логічне підґрунтя для подальшого розгляду практичної реалізації Telegram-боту для бронювання послуг.

РОЗДІЛ 3
ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ TELEGRAM-БОТУ ДЛЯ БРОНЮВАННЯ ПОСЛУГ

3.1 Методика проведення дослідження

Експериментальне дослідження результативності розробленого Telegram-боту для бронювання послуг було проведено з метою перевірки коректності реалізації функціональних можливостей системи, стабільності її роботи та узгодженості збереження даних у реляційній базі даних. Дослідження виконувалося в тестовому середовищі з використанням реального Telegram-клієнта, серверної частини, розгорнутої в хмарній інфраструктурі Amazon Web Services, та бази даних Amazon RDS.

Методика дослідження ґрунтується на сценарному підході, відповідно до якого було сформовано набір типових користувацьких сценаріїв взаємодії з Telegram-ботом. До переліку сценаріїв увійшли: ініціалізація взаємодії за допомогою команди /start, створення бронювання, перегляд власних бронювань, скасування бронювання, обробка спроб повторного бронювання зайнятого часу, обробка некоректних команд, а також перевірка коректності обробки вебхуків у безсерверному середовищі.

Кожен сценарій виконувався вручну через інтерфейс Telegram-боту. Після виконання відповідної дії здійснювалася перевірка стану бази даних, зокрема таблиць bookings та timeslots, з метою підтвердження коректності збереження інформації, оновлення статусів бронювання та зміни доступності часових інтервалів. Такий підхід дозволив оцінити узгодженість роботи користувацького інтерфейсу, серверної бізнес-логіки та рівня зберігання даних.

3.2 Обробка та аналіз отриманих результатів

У ході експериментального дослідження було перевірено всі заплановані функціональні сценарії, результати яких підтверджують коректну роботу розробленого Telegram-боту. Надсилання команди /start ініціювало взаємодію з ботом і повертало користувачу привітальне повідомлення з описом основних можливостей системи, що свідчить про правильну реалізацію початкового етапу взаємодії.

Під час створення бронювання система формувала інтерактивну форму з використанням інлайн-кнопок. Після підтвердження користувачем бронювання відповідний запис зберігався в таблиці bookings бази даних. Аналіз збереженого запису показав коректне збереження ідентифікатора користувача, бізнесу, послуги та відповідного часового інтервалу, а також тексту коментаря, параметрів нагадування та статусу бронювання зі значенням confirmed. Це підтверджує правильну реалізацію бізнес-логіки створення бронювання та збереження користувацьких налаштувань.

Даний процес проілюстровано на рисунку 3.1.

	Row #1
id	b653d81f-75d3-4d2d-af59-221467996217
service_id	a73ae486-766a-4d77-b6e0-a00c3e87030f
start_time	2025-11-09 13:00:00.000 GMT+00:00
end_time	2025-11-09 14:00:00.000 GMT+00:00
is_available	[ ]
created_at	2025-11-08 07:19:19.179 GMT+00:00
updated_at	2025-11-08 15:59:11.063 GMT+00:00

Рисунок 3.1 – Запис бронювання в таблиці bookings після підтвердження користувачем

Подальший аналіз таблиці timeslots засвідчив, що після створення бронювання відповідний часовий інтервал автоматично позначався як недоступний шляхом встановлення поля is_available у значення false. Це забезпечує запобігання повторному бронюванню одного й того самого часу та підтверджує коректну реалізацію механізму управління ресурсами в системі.

Зміна доступності часового інтервалу представлена на рисунку 3.2.

	Row #2
id	a66a9837-ed34-4b68-9a18-93084f5658b8
user_id	1,144,060,743
business_id	cf732c32-4388-4dc4-a7e9-900f2aaa51ab
service_id	a73ae486-766a-4d77-b6e0-a00c3e87030f
timeslot_id	b653d81f-75d3-4d2d-af59-221467996217
comment	тест коментар
wants_reminder	[v]
reminder_minutes_before	60
status	confirmed
created_at	2025-11-08 15:59:11.155 GMT+00:00
updated_at	2025-11-08 15:59:11.155 GMT+00:00

Рисунок 3.2 – Зміна доступності часового інтервалу в таблиці timeslots після створення бронювання

Функціональність перегляду власних бронювань за допомогою команди /bookings працювала коректно, оскільки користувачу відображався список усіх його бронювань із детальною інформацією про послугу, дату, час і статус. Сценарій скасування бронювання також було успішно перевірено: після підтвердження дії відповідний запис оновлювався, а часовий інтервал знову ставав доступним для подальшого бронювання.

При спробі повторного бронювання вже зайнятого часового інтервалу бот коректно повідомляв користувача про конфлікт доступності, не

створюючи дублюючих записів у базі даних. Також підтверджено правильну обробку некоректних або невідомих команд, у відповідь на які бот повертав інформативні повідомлення.

Окремо було перевірено роботу вебхуків. Запити, отримані через Amazon API Gateway, успішно оброблялися AWS Lambda-функцією з інтегрованим NestJS-застосунком, що підтверджує стабільність безсерверної архітектури та коректність маршрутизації повідомлень від платформи Telegram до серверної логіки.

Таким чином, результати експериментального дослідження підтверджують, що розроблений Telegram-бот забезпечує повну реалізацію заявленого функціоналу, коректно взаємодіє з базою даних та демонструє стабільну роботу в хмарному середовищі Amazon Web Services.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було досягнуто поставленої мети та реалізовано всі визначені завдання, що дозволило розробити та дослідити Telegram-бот для бронювання послуг із використанням сучасних хмарних технологій. Отримані результати узагальнюють теоретичні положення та практичні напрацювання, а також підтверджують доцільність обраних архітектурних і технологічних рішень.

У результаті аналізу наукових публікацій і практичних рішень встановлено, що Telegram-боти є ефективним інструментом автоматизації процесів бронювання завдяки високій доступності, простоті інтеграції та підтримці інтерактивних сценаріїв взаємодії. Виявлено тенденцію до використання хмарних і безсерверних архітектур, що забезпечують масштабованість і зниження витрат на інфраструктуру. Отримані результати стали теоретичним підґрунтям для вибору архітектурних і технологічних рішень у подальших етапах роботи.

У ході виконання завдання обґрунтовано вибір фреймворку NestJS для побудови серверної частини, бібліотеки Telegraf для інтеграції з платформою Telegram, а також сервісів Amazon Web Services для розгортання безсерверної архітектури. Обраний технологічний стек забезпечує модульність, масштабованість і надійність системи, що відповідає сучасним вимогам до хмарних інформаційних систем.

У результаті виконання завдання була розроблена реляційна модель бази даних, яка включає взаємопов'язані сутності користувачів, бізнесів, послуг, таймслотів і бронювань. Запропонована структура забезпечує цілісність даних, підтримує складні зв'язки між сутностями та дозволяє ефективно реалізувати бізнес-логіку бронювання і управління доступністю ресурсів.

Було реалізовано Telegram-бот, який підтримує повний цикл бронювання: вибір послуги, дати й часу, введення коментаря, налаштування

нагадування та підтвердження бронювання. Реалізований механізм керування станом діалогу забезпечує послідовність взаємодії та коректну обробку дій користувача. Бот коректно взаємодіє з серверною частиною та базою даних, що підтверджує правильність реалізації програмної логіки.

Інтеграція Telegram-боту з безсерверною інфраструктурою AWS була успішно виконана з використанням Amazon API Gateway та AWS Lambda. Реалізовано обробку вебхуків Telegram у середовищі NestJS із кешуванням обробника, що дозволило зменшити затримки та підвищити продуктивність системи. Отримані результати підтвердили доцільність використання serverless-підходу для подібних застосунків.

Експериментальне тестування підтвердило коректну роботу всіх основних функцій Telegram-боту, включаючи створення, перегляд і скасування бронювань, обробку конфліктів доступності та некоректних команд. Аналіз записів у базі даних засвідчив узгодженість дій користувача з оновленням статусів бронювання та доступності часових інтервалів. Отримані результати свідчать про ефективність і надійність розробленої системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шведа В. О., Повстяна Ю. С. Побудова масштабованої інфраструктури telegram-бота за допомогою AWS-сервісів. Тези доповідей X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві(ІТОНВ-2025) (23-24 травня 2025 року). Луцьк: ЛНТУ, 2025. С. 407-410. URL:https://itonv.lntu.edu.ua/ (дата звернення: 01.06.2025).

2. Evaluating Quality of Chatbots and Intelligent Conversational Agents. URL: https://doi.org/10.48550/arXiv.1704.04579 (дата звернення: 01.06.2025).

3. A Chatbot for Searching and Exploring Open Data: Implementation and Evaluation in E-Government. URL: https://dl.acm.org/doi/ 10.1145/3463677.3463681 (дата звернення: 01.06.2025).

4. Chatbot Development: Framework, Platform, and Assessment Metrics. URL: https://www.researchgate.net/publication/383147845_Chatbot_ Development_Framework_Platform_and_Assessment_Metrics (дата звернення: 05.06.2025).

5. Evaluating Quality of Chatbots and Intelligent Conversational Agents. URL: https://doi.org/10.48550/arXiv.1704.04579 (дата звернення: 12.07.2025).

6. Skyscanner. URL: https://www.skyscanner.net/ (дата звернення: 11.09.2025).

7. Booking. URL: https://www.booking.com/ (дата звернення: 11.09.2025).

8. Uber Eats. URL: https://www.ubereats.com/ (дата звернення: 11.09.2025).

9. H&M. URL: https://www.hm.com/ (дата звернення: 11.09.2025).

10. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. URL: https://www.postgresql.org/docs (дата звернення: 28.09. 2025).

11. NestJS Documentation. NestJS – прогресивний фреймворк для Node.js. URL: https://docs.nestjs.com (дата звернення: 09.09.2025).

12. Amazon Web Services. Amazon API Gateway Developer Guide. URL: https://docs.aws.amazon.com/apigateway (дата звернення: 23.09. 2025).

13. Amazon Web Services. Serverless Architectures with AWS Lambda. URL: https://docs.aws.amazon.com/lambda (дата звернення: 14.09. 2025).

14. Munirov J. J. NESTJS VS EXPRESS.JS: A Comprehensive Comparison. Modern World Education: New Age Problems – New solutions, Vol.2 No.6, 2025. URL: https://incop.org/index.php/mo/article/view/1630 (дата звернення: 14.11. 2025).

15. May A. Top TypeScript ORMs 2025: Comparison of Drizzle, Prisma, Sequelize, TypeORM, MikroORM. Bytebase Blog, 26 May 2025. URL: https://www.bytebase.com/blog/top-typescript-orm/ (дата звернення: 18.11. 2025).