Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

КВАЛІФІКАЦІЙНА РОБОТА
МАГІСТРА
за спеціальністю 123 «Комп'ютерна інженерія»

Тема наукової роботи: МЕТОДИ АВТОМАТИЗАЦІЇ КЛІЄНТСЬКОЇ ПІДТРИМКИ НА ОСНОВІ ЧАТ-БОТА

Виконав	____	Є. В. Сердюк
Керівник роботи	____	Ю. О. Кумченко
Нормоконтроль	____	Д. І. Кузнєцов
Завідувач кафедри	____	А. І. Купін

Кривий Ріг 2025

Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерних систем та мереж

Ступінь вищої освіти магістр
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри, голова циклової комісії

_____А. І. Купін

"___" _____20___року

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Сердюк Євгеній Володиимирович
(прізвище, ім'я, по батькові)

1.Тема роботи Методи автоматиизації клієнської підтриимки на основі чат бота

керівник роботи_____,
затверджені наказом вищого навчального закладу від "____"_____20__року№____
2. Строк подання студентом роботи_____
3. Вихідні дані до роботи_____

4. Зміст розрахунково-пояснювальної записки (перелік питань,які потрібно розробити)
Дослідити історію автоатиизації клієнської підтр.
Розробити теоретичну модель роботи чат-бота
Розробити готові ситсеми автоматиизації за 2 етодами
Виконати порівняння та аналіз ефектиивності використаних методів

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
Презентація PowerPoint

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Огляд інформації щодо теми	10.09.25 – 12.09.25	Виконав
2	Сортування знайденої інформації	13.09.25 – 20.09.25	Виконав
3	Огляд історії автоматиизації	21.09.25 – 25.09.25	Виконав
4	Написання першого розділу	26.09.25 – 30.09.25	Виконав
5	Напиисання другого розділу	01.10.25 – 07.10.25	Виконав
6	Написання коду	08.10.25 – 16.10.25	Виконав
7	Написання оформлення третього розділу	26.10.25 – 05.11.25	Виконав
8	Написання висновків	06.11.25 – 15.11.25	Виконав
9	Оформлення роботи	16.11.25 – 30.11.25	Виконав

Студент _____ _____
(підпис) (прізвище та ініцали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка: 108 сторінок, 63 рисунків, 10 таблиць, 2 додатків, 23 використаних джерел.

Об’єкт дослідження – процеси автоматизації клієнтської підтримки в сервісних системах та інформаційних середовищах, включаючи методи створення, організації та інтеграції чат-ботів у бізнес-процеси.

Першиий розділ присвячено теоретичному аналізу проблеми автоматизації клієнтської підтримки, розглянуто роль чат-ботів у сучасних інформаційних системах, огляд еволюції чат-ботів, їх класифікаційних моделей, архітектур, принципів роботи. Другиий розділ присвячено технічнічним аспектам побудови системи чат-бота. Проведено порівняння мов програмування та технологій, обґрунтовано вибір Python як основного інструмента розробки. Третій розділ приисвячено виконанню практичної реалізації двох версій чат-бота: програмної, створеної мовою Python із використанням Telegram API та SQLite, і no-code реалізації, побудованої у середовищі BotHelp.

КЛЮЧОВІ СЛОВА: ЧАТ-БОТ, АВТОМАТИЗАЦІЯ, КЛІЄНТСЬКА ПІДТРИМКА, PYTHON, BOTHELP, NLP, API, TELEGRAM API, FSM, БАЗА ЗНАНЬ, ОБРОБКА ПРИРОДНОЇ МОВИ, ЗАЯВКА, СИСТЕМА ПІДТРИМКИ.

EXPLANATORY NOTE

Explanatory note: 108 pages, 63 figures, 10 tables, 2 appendices,
23 sources used.

The object of the study is the processes of customer support automation in service systems and information environments, including methods of creating, organizing and integrating chatbots into business processes.

The first section is devoted to the theoretical analysis of the problem of customer support automation, the role of chatbots in modern information systems is considered, an overview of the evolution of chatbots, their classification models, architectures, and principles of operation is reviewed. The second section is devoted to the technical aspects of building a chatbot system. A comparison of programming languages and technologies is made, and the choice of Python as the main development tool is justified. The third section is devoted to the practical implementation of two versions of the chatbot: a software version created in Python using the Telegram API and SQLite, and a no-code implementation built in the BotHelp environment.

KEYWORDS: CHAT BOT, AUTOMATION, CUSTOMER SUPPORT, PYTHON, BOTHELP, NLP, API, TELEGRAM API, FSM, KNOWLEDGE BASE, NATURAL LANGUAGE PROCESSING, APPLICATION, SUPPORT SYSTEM.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

1. **AI (Artificial Intelligence)** — штучний інтелект; галузь інформатики, що займається створенням систем, здатних виконувати завдання, які потребують інтелектуальних здібностей людини.
2. **ML (Machine Learning) —** машинне навчання; підгалузь штучного інтелекту, що передбачає навчання систем на основі даних без явного програмування.
3. **NLP (Natural Language Processing) —** обробка природної мови; напрям штучного інтелекту, який дозволяє системам розуміти, аналізувати й генерувати людську мову.
4. **API (Application Programming Interface) —** програмний інтерфейс прикладного програмування; набір інструментів і протоколів для взаємодії між різними програмами.
5. **JSON (JavaScript Object Notation) —** текстовий формат обміну даними, що використовується для передачі структурованої інформації між клієнтом і сервером.
6. **HTTP (HyperText Transfer Protocol) —** протокол передачі гіпертексту, що використовується для обміну даними в мережі Інтернет.
7. **UI (User Interface) —** користувацький інтерфейс; сукупність засобів, через які користувач взаємодіє із системою.
8. **UX (User Experience) —** користувацький досвід; сукупність вражень користувача від взаємодії з програмним продуктом.
9. **DB (Database) —** база даних; структурована сукупність даних, що зберігаються та керуються за допомогою систем управління базами даних.
10. **SQL (Structured Query Language) —** мова структурованих запитів, призначена для створення, зміни та обробки даних у реляційних базах даних.

ВСТУП

У сучасному цифровому середовищі стрімкий розвиток інформаційних технологій суттєво трансформує способи взаємодії між організаціями та користувачами. Активна диджиталізація бізнес-процесів, широке впровадження електронних сервісів, розширення можливостей мережі Інтернет та переорієнтація компаній на онлайн-комунікації зумовлюють підвищений попит на ефективні, швидкі та автоматизовані системи підтримки клієнтів. У таких умовах одним із найперспективніших інструментів оптимізації комунікацій є чат-боти — програмні системи, які застосовують методи штучного інтелекту, обробки природної мови (NLP) та алгоритмічних моделей для забезпечення автоматизованого діалогу з користувачем.

Завдяки широким можливостям інтеграції з базами знань, CRM-системами, аналітичними платформами та зовнішніми сервісами, чат-боти сьогодні використовуються у банківській сфері, електронній комерції, медицині, освіті, службах доставки, технічній підтримці та держсекторі. Вони дозволяють суттєво підвищити швидкість обробки звернень, забезпечити цілодобову підтримку без участі оператора, знизити навантаження на персонал та покращити якість взаємодії з користувачами.

Хоча існує велика кількість готових платформ для створення чат-ботів, проблема ефективної автоматизації клієнтської підтримки залишається актуальною. Типові боти часто працюють на основі простих сценаріїв і не здатні гнучко реагувати на різноманітні запити або адаптуватися до контексту діалогу. Інтеграція складних ботів у корпоративну інфраструктуру потребує ґрунтовних технічних знань, продуманої архітектури та використання сучасних методів машинного навчання та семантичного аналізу.

Важливою складовою розвитку сучасних чат-ботів є впровадження моделей обробки природної мови, механізмів самонавчання, а також гібридних підходів, що поєднують традиційні правила формування відповідей з можливостями нейронних мереж. Ефективні чат-боти дозволяють автоматизувати клієнтську підтримку, скоротити час обробки звернень, зменшити кількість помилок та підвищити рівень задоволеності користувачів.

Актуальність роботи. Актуальність теми полягає у зростаючій потребі організацій у впровадженні інтелектуальних систем підтримки, здатних автоматично обробляти звернення користувачів у зручний, швидкий та ефективний спосіб. Використання чат-ботів дозволяє забезпечити безперервний сервіс, знижує витрати на обслуговування клієнтів та створює основу для підвищення якості цифрової взаємодії.

Сучасні тенденції розвитку штучного інтелекту, NLP-моделей та платформ автоматизації відкривають нові можливості для створення адаптивних чат-ботів, які здатні розуміти контекст, обробляти варіативні запити та взаємодіяти з інформаційними системами підприємства. У рамках кваліфікаційної роботи досліджується як програмна, так і no-code реалізація чат-бота, що забезпечує комплексне вивчення та порівняння технологічних підходів.

Мета і завдання дослідження. Розробити та проаналізувати методи автоматизації клієнтської підтримки на основі чат-бота, створити функціональний програмний прототип та no-code реалізацію, оцінити їх ефективність та визначити перспективи використання в реальних умовах.

Основними завданнями дослідження є проведення аналізу сучасних технологій і підходів до побудови чат-ботів; дослідити можливості Python та no-code платформ у реалізації автоматизованих систем підтримки; розробити архітектуру чат-бота з використанням механізмів FSM, NLP, баз даних та API-інтеграцій; реалізувати програмний чат-бот у середовищі Python; створити аналогічну модель чат-бота на платформі BotHelp; виконати тестування, порівняння функціональних можливостей та оцінку продуктивності обох реалізацій.

Об’єкт дослідження. Процеси автоматизації клієнтської підтримки в інформаційних системах із застосуванням чат-ботів.

Предмет дослідження. Методи, моделі та програмні засоби побудови чат-ботів, алгоритми обробки природної мови, бази знань та механізми інтеграції з зовнішніми сервісами.

Методи дослідження. У роботі використано: системний аналіз програмних систем та інструментів автоматизації; метод порівняння технологічних платформ; аналітичні методи дослідження структури діалогів; експериментальне тестування реалізованих чат-ботів; моделювання логіки роботи системи за допомогою кінцевих автоматів.

Наукова новизна одержаних результатів. Удосконалено підхід до побудови систем клієнтської підтримки шляхом поєднання програмної та no-code реалізацій чат-ботів, розроблено модель логіки на основі FSM з можливістю адаптації під різні сценарії, а також сформовано порівняльний аналіз ефективності альтернативних технологічних рішень у контексті автоматизації сервісних процесів.

Практичне значення отриманих результатів. Результати дослідження можуть бути використані: для впровадження систем автоматизованої підтримки в сервісних центрах; у проєктуванні чат-ботів для комерційних, освітніх та державних структур; при розробці універсальних рішень для обробки клієнтських звернень та формування баз знань. Запропоновані підходи забезпечують можливість створення масштабованих, адаптивних і високопродуктивних систем клієнтської підтримки.

Апробація роботи. Апробація результатів дослідження здійснювалася в рамках XVIII Всеукраїнської науково-практичної WEB-конференції аспірантів, студентів та молодих учених «Комп'ютерні інтелектуальні системи та мережі» (CISN-2025), що проводилася на базі Криворізького національного університету. Під час конференції були представлені основні положення, методологія та результати кваліфікаційної роботи, а також проведено їх обговорення в науковому середовищі.

1. ТЕОРЕТИЧНІ ОСНОВИ АВТОМАТИЗАЦІЇ КЛІЄНТСЬКОЇ ПІДТРИМКИ

1.1. Сутність та роль клієнтської підтримки в сучасних інформаційних системах

Клієнтська підтримка є однією з ключових складових будь-якої сучасної організації, яка взаємодіє зі споживачами товарів чи послуг. Вона забезпечує комунікацію між компанією та користувачем, сприяє підвищенню задоволеності клієнтів, формуванню позитивного іміджу бренду та зміцненню довгострокових відносин із клієнтською базою. У сучасних умовах високої конкуренції рівень обслуговування стає одним із визначальних чинників успіху підприємства поряд із якістю продукту та ціною.

Клієнтська підтримка — це сукупність організаційних, технічних і програмних засобів, спрямованих на забезпечення користувачів необхідною інформацією, допомогою у вирішенні технічних проблем та супроводом у процесі використання продукту чи послуги. Її основними завданнями є своєчасна реакція на звернення клієнтів, оперативне усунення проблем, надання консультацій і рекомендацій, а також збір зворотного зв’язку для подальшого вдосконалення сервісу.

З розвитком інформаційних технологій відбулися значні зміни в підходах до організації клієнтської підтримки. Якщо раніше основним каналом комунікації був телефонний дзвінок або електронна пошта, то сьогодні значну частину звернень обробляють через цифрові платформи — веб-сайти, мобільні застосунки, соціальні мережі та месенджери. У результаті клієнтська підтримка перетворилася на багатоканальну систему взаємодії, де користувач може звертатися у зручний для нього спосіб і в будь-який час.

У сучасних інформаційних системах клієнтська підтримка є невід’ємним компонентом загальної IT-інфраструктури підприємства. Вона інтегрується з базами даних клієнтів (CRM-системами), аналітичними платформами, системами управління контентом (CMS) і маркетинговими інструментами. Така інтеграція дозволяє не лише оперативно вирішувати проблеми користувачів, а й формувати аналітичні звіти, відстежувати частоту звернень, виявляти типові проблеми, прогнозувати навантаження на службу підтримки[1].

Завдяки інформаційним технологіям клієнтська підтримка переходить від реактивного підходу (коли оператор реагує на вже отримане звернення) до проактивного підходу, який передбачає передбачення потреб користувача. Наприклад, система може автоматично пропонувати рішення ще до того, як клієнт

сформулює запит, або надсилати повідомлення з порадами на основі його попередніх дій.

Разом із тим, збільшення обсягу звернень і розширення каналів комунікації створює нові виклики. Людські ресурси мають обмеження: оператори не здатні обробляти одночасно велику кількість запитів, а робота служби підтримки в режимі 24/7 потребує значних фінансових витрат. Саме тому виникає необхідність автоматизації процесів клієнтської підтримки, що дозволяє оптимізувати роботу персоналу та забезпечити постійний контакт із клієнтом.

Одним із найбільш ефективних шляхів вирішення цієї проблеми є впровадження інтелектуальних систем підтримки на основі штучного інтелекту (AI) і технологій обробки природної мови (NLP). Такі системи здатні не лише відповідати на типові запитання, але й аналізувати контекст повідомлень, навчатися на основі попередніх діалогів, адаптуватися до специфіки конкретного бізнесу. Найпоширенішою формою реалізації цих систем є чат-боти — автоматизовані програми, які імітують спілкування з людиною через текстовий або голосовий інтерфейс.

Роль клієнтської підтримки сьогодні виходить за межі простої допомоги у вирішенні технічних проблем. Вона стає стратегічним інструментом взаємодії з користувачем, який впливає на рівень довіри, повторні покупки, утримання клієнтів і формування лояльності. Підтримка, що ґрунтується на сучасних інформаційних системах, здатна перетворити процес обслуговування на джерело цінних даних для бізнес-аналітики та вдосконалення продукту.

Таким чином, клієнтська підтримка у сучасних інформаційних системах — це складна багаторівнева структура, що поєднує технічні засоби, програмне забезпечення та людський фактор. Її ефективність визначається здатністю швидко та точно задовольняти інформаційні потреби користувачів. У цьому контексті автоматизація процесів підтримки набуває ключового значення, оскільки дозволяє підвищити якість обслуговування, знизити витрати й забезпечити постійну доступність сервісів.

Отже, розвиток клієнтської підтримки тісно пов’язаний із розвитком інформаційних технологій, зокрема систем штучного інтелекту. Використання інтелектуальних агентів і чат-ботів є логічним етапом еволюції цієї сфери, який відкриває нові можливості для побудови ефективних і масштабованих систем взаємодії з користувачами[1].

1.2. Методи автоматизації процесів обслуговування користувачів

Автоматизація процесів обслуговування користувачів є одним із ключових напрямів цифрової трансформації підприємств. Вона передбачає впровадження технологічних рішень, що дозволяють мінімізувати участь людини у рутинних операціях, підвищити швидкість реагування на запити клієнтів і забезпечити стабільну якість сервісу. Такий підхід не лише оптимізує роботу контакт-центрів,

а й сприяє підвищенню загальної ефективності бізнес-процесів, дозволяючи персоналу зосередитися на виконанні більш складних і творчих завдань.

Зі зростанням кількості цифрових каналів комунікації — вебсайтів, мобільних додатків, месенджерів, соціальних мереж — обсяг звернень користувачів постійно збільшується. Це створює необхідність використання інтелектуальних методів обробки запитів, здатних забезпечити масштабованість системи підтримки без пропорційного зростання кількості операторів. У цьому контексті автоматизація виступає стратегічним інструментом, який дозволяє організаціям підтримувати високий рівень клієнтського сервісу навіть за умов зростання навантаження.

Сучасні інформаційні системи клієнтської підтримки інтегрують різноманітні механізми автоматизації — від простих форм обробки заявок до складних систем, що використовують штучний інтелект (AI), машинне навчання (ML) та обробку природної мови (NLP). Такі технології дозволяють не лише реагувати на запити користувачів, але й прогнозувати їхні потреби, пропонуючи релевантні рішення ще до виникнення проблеми. Це створює умови для проактивного обслуговування, яке підвищує задоволеність клієнтів і лояльність до бренду.

Одним із найважливіших напрямів автоматизації є використання чат-ботів та віртуальних асистентів, здатних здійснювати первинну обробку звернень, проводити класифікацію запитів, надавати довідкову інформацію чи перенаправляти користувача до відповідного фахівця. Використання таких інструментів дозволяє скоротити час відповіді, забезпечити цілодобову доступність сервісу та значно зменшити операційні витрати. Згідно з дослідженнями Gartner, до 2025 року понад 70% компаній у сфері електронної комерції та послуг впровадять чат-боти для підтримки клієнтів, що підтверджує стратегічну важливість цього напрямку.

Важливим аспектом автоматизації є інтеграція клієнтських сервісів у єдину цифрову екосистему підприємства. Це передбачає зв’язок між CRM-системами, базами даних, аналітичними модулями та каналами комунікації. Такий підхід забезпечує повну видимість історії взаємодії з клієнтом, що дозволяє підвищити точність відповідей, персоналізувати обслуговування та швидше вирішувати проблеми.

Окрім цього, автоматизація процесів підтримки користувачів має вагоме значення для аналітики клієнтського досвіду (Customer Experience Analytics). Системи здатні збирати, структурувати та аналізувати великі обсяги даних про звернення, типові проблеми, емоційний тон спілкування тощо. На основі цих даних компанії можуть визначати слабкі місця у сервісі, прогнозувати навантаження на операторів та приймати обґрунтовані управлінські рішення.

Не менш важливим є аспект гнучкості та масштабованості автоматизованих систем. Хмарні технології, мікросервісна архітектура та платформи з відкритими

інтерфейсами (API) дозволяють швидко адаптувати систему під зміни бізнес-процесів, розширювати її функціональність або інтегрувати нові канали комунікації. Таким чином, автоматизація не лише підвищує ефективність поточних операцій, але й забезпечує довгострокову адаптивність компанії до ринкових змін.

У підсумку, автоматизація процесів клієнтської підтримки є не просто технічним оновленням, а комплексною стратегією цифрового розвитку підприємства. Вона поєднує інноваційні технології, бізнес-аналітику та орієнтацію на потреби користувача, формуючи основу для побудови сучасної, ефективної та клієнтоорієнтованої інформаційної системи.

1.2.1 Класифікація методів автоматизації обслуговування

Методи автоматизації клієнтської підтримки можна умовно поділити на кілька груп залежно від ступеня інтелектуальності та рівня взаємодії з користувачем:

Правилові системи (rule-based systems) — базуються на заздалегідь визначених сценаріях і наборах правил. Такі системи реагують на конкретні ключові слова або фрази та надають відповідь відповідно до заздалегідь заданої логіки[3].

Наприклад, якщо користувач вводить «забув пароль», система автоматично пропонує інструкцію з відновлення доступу.Перевагами цього підходу є простота реалізації й передбачуваність результатів. Однак недоліком є обмежена гнучкість, оскільки такі системи не здатні інтерпретувати контекст і розуміти різноманіття формулювань запитів.

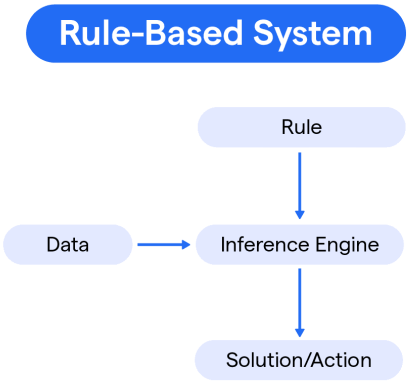


Рисунок 1.1 - Системи на основі правил (rule-based systems)

Системи на основі пошуку (retrieval-based systems) — працюють за принципом пошуку найбільш релевантної відповіді серед заздалегідь підготовленої бази знань. Вони можуть використовувати методи обробки природної мови (NLP), такі як лематизація, аналіз синонімів, обчислення семантичної схожості текстів. Перевага таких систем — у можливості надання

точних і коректних відповідей без необхідності повного розуміння контексту. Недолік — складність підтримки актуальності бази знань і неможливість формувати нові відповіді.

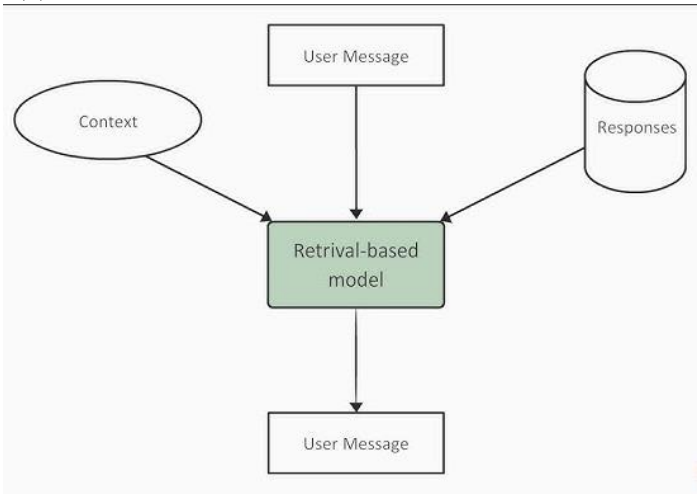


Рисунок 1.2 - Системи на основі пошуку (retrieval-based systems)

Генеративні системи (generative systems) — ґрунтуються на алгоритмах машинного навчання, зокрема нейронних мережах і мовних моделях. Вони не просто шукають готову відповідь, а генерують її самостійно на основі навчальних даних.

Такі моделі можуть розуміти контекст діалогу, визначати інтенцію користувача та підтримувати довготривалу логіку спілкування. Їхнім прикладом є сучасні чат-боти, що працюють на основі трансформерних архітектур, таких як GPT, BERT або LLaMA. Головною перевагою генеративних систем є адаптивність і здатність до навчання, однак вони вимагають великих обсягів даних і обчислювальних ресурсів[2].

Гібридні системи — поєднують сценарійно-правилові підходи з елементами штучного інтелекту. Такий підхід дозволяє досягти балансу між передбачуваністю і гнучкістю. Наприклад, базові запити обробляються за фіксованими сценаріями, а складні — передаються генеративній моделі або оператору.

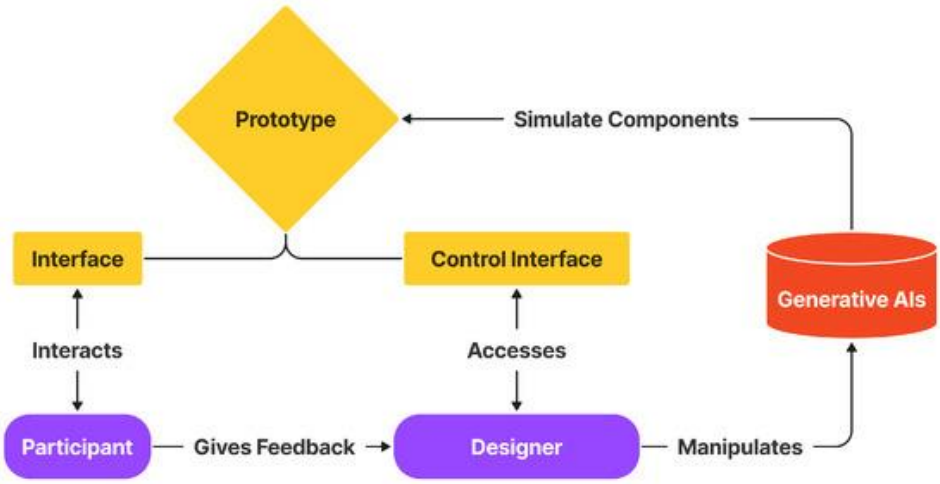


Рисунок 1.3 - Генеративні системи (generative systems)

1.2.2 Методи автоматизації на основі штучного інтелекту

Використання штучного інтелекту у клієнтській підтримці базується на кількох основних технологічних напрямках:

Обробка природної мови (Natural Language Processing, NLP) — дозволяє системі “розуміти” запити користувача, виділяти ключові сутності, визначати намір (intent) і формувати відповідь. Серед типових методів NLP: токенізація, частиномовний аналіз (POS-tagging), визначення сутностей (Named Entity Recognition), семантичне кодування та класифікація тексту.

Машинне навчання (Machine Learning) — використовується для аналізу історії звернень і вдосконалення системи на основі отриманих даних. Зокрема, моделі класифікації допомагають визначити тип запиту (наприклад, «технічна підтримка», «питання про оплату», «загальна інформація»), а моделі рекомендацій — пропонують користувачу оптимальні рішення.

Системи розпізнавання голосу (Speech-to-Text, Text-to-Speech) — дозволяють реалізувати голосову взаємодію між користувачем і системою. Це особливо актуально для мобільних застосунків і контакт-центрів.

Аналіз настроїв (Sentiment Analysis) — застосовується для оцінки емоційного стану користувача під час спілкування. Система може реагувати на негативний тон повідомлення, підвищуючи рівень уваги або передаючи звернення оператору.

1.2.3 Інструменти та середовища реалізації

Сучасні середовища розробки дозволяють створювати автоматизовані системи клієнтської підтримки без необхідності будувати все “з нуля”. Завдяки вже готовим інструментам для обробки природної мови, інтеграції з месенджерами та підключення до зовнішніх сервісів, процес розробки стає значно швидшим і гнучкішим. Такі платформи надають розробникам інтерфейси для

навчання моделей, побудови сценаріїв діалогу, аналітики звернень та управління контентом.

Найпопулярнішими рішеннями для створення чат-ботів і систем підтримки є:

1. Dialogflow (Google). Це одна з найпоширеніших платформ для створення інтелектуальних чат-ботів із вбудованими можливостями Natural Language Processing (NLP). Вона дозволяє розпізнавати наміри користувача, обробляти контекст діалогу та формувати гнучкі сценарії спілкування. Основні переваги:

підтримка багатьох каналів комунікації — Telegram, Facebook Messenger, WhatsApp, Viber, Slack тощо;

глибока інтеграція з Google Cloud Platform, що забезпечує масштабованість і надійність;

можливість створення мультимовних ботів;

наявність зручного вебінтерфейсу для адміністрування без необхідності глибоких знань програмування. Dialogflow ідеально підходить для створення корпоративних ботів, що потребують високої стабільності й підтримки декількох мов.

2. Microsoft Bot Framework. Це потужна платформа від компанії Microsoft, орієнтована на розробку масштабованих корпоративних рішень. Вона надає можливість створювати складні діалогові системи з підтримкою різних каналів зв’язку — від сайтів і месенджерів до голосових інтерфейсів. Основні переваги:

інтеграція з екосистемою Azure, що дозволяє використовувати сервіси штучного інтелекту, аналітики та хмарного зберігання;

сумісність із бізнес-системами Microsoft (Dynamics 365, Teams, Outlook);

можливість використання модулів для автоматичного перекладу, розпізнавання мови та аналізу настрою користувача. Microsoft Bot Framework переважно застосовується у великих організаціях, де чат-бот є частиною загальної IT-інфраструктури.

3. Rasa. Це платформа з відкритим кодом (open source), що дозволяє створювати AI-базовані чат-боти з високим рівнем кастомізації. Вона складається з двох основних компонентів:

Rasa NLU — відповідає за розпізнавання інтенцій і вилучення сутностей;

Rasa Core — забезпечує управління діалогом і логікою взаємодії. Основні переваги Rasa:

повна контрольованість даних (бот може працювати локально без використання сторонніх хмарних сервісів);

можливість навчання на власних наборах даних;

підтримка складних сценаріїв і контекстних діалогів;

інтеграція з Python та сторонніми бібліотеками ML/NLP. Rasa найчастіше використовується у проєктах, де важлива безпека даних, гнучкість налаштувань і можливість автономної роботи.

4. Telegram Bot API. Telegram надає простий і зручний інтерфейс для створення ботів безпосередньо у своєму месенджері. Завдяки Bot API розробники можуть швидко створювати прототипи систем клієнтської підтримки, отримуючи доступ до великої аудиторії користувачів. Основні переваги:

простота реалізації через HTTP-запити або Python-бібліотеки (наприклад, aiogram, pyTelegramBotAPI);

підтримка форматування повідомлень, кнопок, меню та інтерактивних елементів;

можливість підключення до зовнішніх API та баз даних. Це рішення оптимальне для малого та середнього бізнесу, який потребує швидкого запуску ефективного бота без складної інфраструктури.

5. Python-бібліотеки для NLP та ML. Для створення інтелектуальних модулів, що забезпечують аналіз текстів і навчання моделей, активно застосовуються бібліотеки:

NLTK (Natural Language Toolkit) — базовий інструмент для обробки тексту, токенізації, лематизації, побудови частотних словників;

spaCy — високопродуктивна бібліотека для лінгвістичного аналізу, виявлення сутностей (NER) та визначення частин мови;

Transformers (Hugging Face) — набір сучасних попередньо навчених моделей (GPT, BERT, RoBERTa), які використовуються для семантичного аналізу запитів;

TensorFlow / PyTorch — фреймворки машинного навчання, що дозволяють створювати та навчати нейронні мережі для класифікації текстів і прогнозування намірів.

Поєднання цих інструментів створює гнучку екосистему для розробки чат-ботів будь-якого рівня складності — від простих FAQ-рішень до повноцінних інтелектуальних систем клієнтської підтримки, інтегрованих із базами даних, CRM або зовнішніми API[7].

1.2.4 Переваги автоматизації клієнтської підтримки

Запровадження методів автоматизації в системах обслуговування клієнтів забезпечує низку суттєвих переваг, які безпосередньо впливають на ефективність роботи підприємства та якість комунікації з користувачами. Автоматизація дає змогу оптимізувати процеси обробки звернень, зменшити кількість помилок, підвищити швидкість реагування та забезпечити безперервність обслуговування.

Скорочення часу обробки звернень. Автоматизовані системи, зокрема чат-боти, дозволяють значно зменшити час відповіді на запити клієнтів. Якщо раніше користувачам доводилося чекати з’єднання з оператором, то тепер система

миттєво аналізує запит і надає відповідь на основі бази знань або попередньо визначених сценаріїв. Це скорочує середній час обробки звернення з кількох хвилин до кількох секунд.

Зменшення навантаження на персонал. Автоматизація звільняє операторів від рутинних, однотипних завдань, дозволяючи зосередитись на складніших або нестандартних ситуаціях. Як результат, підвищується продуктивність працівників і зменшується потреба у великій кількості фахівців підтримки. Це також дає можливість підприємству раціональніше використовувати людські ресурси та знизити витрати на утримання персоналу.

Робота системи у режимі 24/7. На відміну від людських операторів, автоматизовані рішення функціонують цілодобово, без вихідних і святкових днів. Така безперервна доступність особливо важлива для компаній, які працюють на міжнародному ринку або мають широку клієнтську базу. Завдяки цьому користувач може отримати допомогу у будь-який час, незалежно від робочого графіка служби підтримки.

Підвищення точності та узгодженості відповідей. Людський фактор часто призводить до неточностей або різних трактувань однієї й тієї ж інформації. Натомість автоматизовані системи працюють за заздалегідь визначеними алгоритмами, що забезпечує єдині стандарти комунікації та високу точність наданих відповідей. Це підвищує рівень довіри користувачів до сервісу й формує позитивний імідж компанії.

Збирання та аналітика даних про запити користувачів. Сучасні автоматизовані системи здатні фіксувати всі звернення, класифікувати їх за темами, виявляти найпоширеніші питання та формувати статистику. Отримані дані використовуються для аналітики та вдосконалення бізнес-процесів: розширення бази знань, оптимізації FAQ-розділів, підготовки персоналу або розробки нових послуг. Таким чином, автоматизація не лише обслуговує користувачів, а й виступає важливим інструментом збору інформації для стратегічного управління.

Крім того, автоматизація сприяє підвищенню задоволеності клієнтів, адже користувач отримує швидку, чітку та точну відповідь без необхідності очікування оператора. Завдяки зменшенню часу реагування та стабільності комунікації формується відчуття надійності та зручності сервісу. У довгостроковій перспективі це позитивно впливає на лояльність клієнтів, збільшує рівень повторних звернень і покращує репутацію компанії на ринку.

1.3. Чат-боти як інструмент автоматизації клієнтської підтримки

У сучасних умовах цифровізації бізнес-процесів чат-боти стали одним із найефективніших засобів автоматизації клієнтської підтримки. Вони дозволяють організаціям забезпечувати постійну взаємодію з користувачами, швидко обробляти запити та знижувати навантаження на операторів контакт-центрів.

Завдяки розвитку штучного інтелекту, машинного навчання та технологій обробки природної мови (NLP), чат-боти перетворилися з простих сценарійних програм у складні інтелектуальні системи комунікації, здатні імітувати природну мову спілкування людини та адаптуватися до контексту діалогу.

Чат-боти відіграють важливу роль у сучасній концепції цифрової взаємодії між компанією та клієнтом. Вони дозволяють автоматизувати типові процеси обслуговування: консультування, реєстрацію звернень, обробку замовлень, інформування про статус послуг чи товарів, бронювання, надання технічної допомоги тощо. Використання таких систем дає змогу скоротити час очікування відповіді, зменшити витрати на утримання персоналу й забезпечити цілодобову доступність сервісу.

Сучасні чат-боти базуються на поєднанні алгоритмів розпізнавання намірів (intent recognition), аналізу контексту діалогу та генерації природних відповідей. Їх ефективність забезпечується за рахунок впровадження технологій машинного навчання (ML), які дозволяють системі вдосконалювати свою поведінку на основі попередніх взаємодій із користувачами. Використання методів глибокого навчання (Deep Learning) та нейронних мереж розширює можливості бота щодо аналізу семантики повідомлень, розуміння емоційного стану користувача та формування релевантних відповідей навіть у нестандартних ситуаціях.

Ключовим компонентом інтелектуального чат-бота є модуль обробки природної мови (NLP), який перетворює текстові або голосові запити у структуровану форму, зрозумілу для системи. За допомогою таких технологій, як токенізація, лематизація, аналіз частин мови (POS-tagging) і визначення сутностей (Named Entity Recognition), бот здатен “розуміти” людські висловлювання у природній формі. Розвиток мовних моделей нового покоління, зокрема BERT, GPT та T5, забезпечив якісно новий рівень діалогових можливостей — від розпізнавання контексту розмови до генерації логічних, зв’язних і персоналізованих відповідей[5].

Ще однією перевагою сучасних чат-ботів є їх інтеграційна гнучкість. Використання API (Application Programming Interface) дозволяє підключати ботів до внутрішніх систем підприємства — CRM, ERP, баз даних, білінгових сервісів, поштових і месенджерних платформ. Це дає змогу створювати єдине інформаційне середовище, у якому клієнт може отримати комплексну допомогу без залучення додаткових ресурсів.

Значний внесок у розвиток технології зробили хмарні сервіси (Google Dialogflow, Amazon Lex, Microsoft Azure Bot Service), які надали розробникам готові модулі для NLP, аналітики та масштабування. Завдяки цьому навіть невеликі компанії можуть розгорнути ефективну систему підтримки без потреби у великих інвестиціях в інфраструктуру.

Важливим напрямом розвитку є також поєднання чат-ботів із системами аналітики даних та обробки зворотного зв’язку. Такі інтеграції дозволяють

збирати статистику звернень, відстежувати типові проблеми клієнтів, визначати рівень задоволеності користувачів і на основі цього покращувати якість сервісу. У перспективі чат-боти можуть виступати не лише як інструмент комунікації, а й як аналітичний елемент бізнес-процесів, що допомагає приймати управлінські рішення.

Таким чином, чат-боти стають невід'ємною частиною цифрової екосистеми сучасного підприємства. Вони поєднують у собі функції консультаційного агента, аналітичного інструмента та каналу комунікації. Використання технологій штучного інтелекту й NLP забезпечує високий рівень автоматизації, персоналізації та ефективності клієнтської підтримки, що в кінцевому підсумку сприяє підвищенню конкурентоспроможності організацій у цифрову епоху.

1.3.1 Історія розвитку чат-ботів

Перші спроби створення систем, здатних вести діалог із користувачем, з'явилися ще в 1960-х роках. Однією з перших таких систем була ELIZA, розроблена Джозефом Вайценбаумом у Массачусетському технологічному інституті (MIT) у 1966 році. ELIZA імітувала спілкування психотерапевта, відповідаючи на фрази користувача за допомогою набору шаблонів і правил.

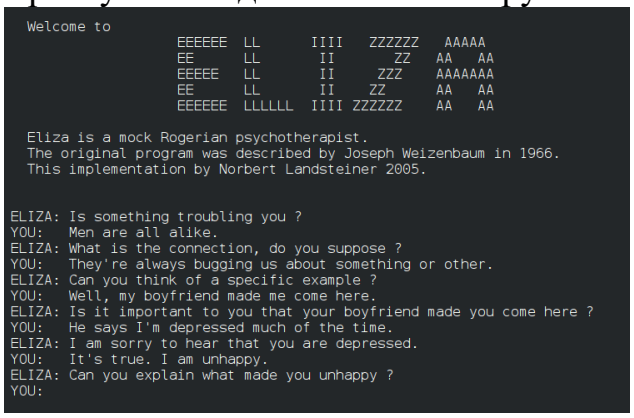


Рисунок 1.4 – Система ELIZA

У 1970-х роках з'явилася програма PARRY, яка вже мала елементи моделювання психічного стану співрозмовника. Подальший розвиток чат-ботів був пов'язаний із появою персональних комп'ютерів та мережевих технологій.

На початку 2000-х років чат-боти активно використовувались у месенджерах ICQ, MSN Messenger та Skype. Проте їхня функціональність залишалася обмеженою — вони реагували лише на чітко визначені команди.

Справжній прорив відбувся після впровадження технологій машинного навчання та NLP. У 2016 році Facebook, Telegram, Microsoft і Google відкрили публічні платформи для створення ботів, що дало поштовх масовому розвитку інтелектуальних агентів. Сьогодні чат-боти використовуються не лише для клієнтської підтримки, а й у сферах електронної комерції, освіти, медицини, банківських послуг і навіть у державному секторі.

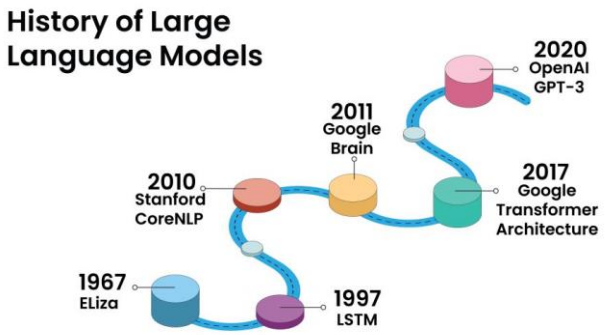


Рисунок 1.5 – Історія LLM

1.3.2 Принцип роботи чат-ботів

Типовий чат-бот є програмним агентом, що взаємодіє з користувачем через текстовий або голосовий інтерфейс. Його основне завдання — розпізнати намір користувача (intent) і надати релевантну відповідь або виконати дію.

Загальна архітектура чат-бота зазвичай включає такі основні компоненти:

Інтерфейс користувача (User Interface) — забезпечує взаємодію між користувачем і системою. Це може бути месенджер (Telegram, Viber, Facebook Messenger), веб-чат на сайті або голосовий інтерфейс.

Модуль обробки природної мови (NLP Engine) — відповідає за розпізнавання введеного тексту, аналіз граматичних конструкцій, визначення ключових слів і наміру користувача. На цьому етапі застосовуються алгоритми токенізації, лематизації, векторизації та класифікації тексту.

База знань або сценарій діалогу (Knowledge Base / Dialogue Flow) — містить правила, шаблони або дані, що визначають, як бот має реагувати на конкретні запити.

Модуль прийняття рішень (Decision Logic) — визначає найкращу відповідь або дію на основі контексту розмови. У більш складних системах він може використовувати методи машинного навчання або експертні системи.

Модуль інтеграції (Integration Layer) — забезпечує взаємодію з зовнішніми сервісами, базами даних чи CRM-системами. Це дозволяє чат-боту виконувати реальні дії: створювати заявки, бронювати послуги, перевіряти статус замовлення тощо.

1.3.3 Типи чат-ботів за рівнем інтелектуальності

Чат-боти можна класифікувати за рівнем складності та застосовуваними методами:

Сценарійні (rule-based) — працюють за принципом попередньо визначених діалогів. Вони підходять для виконання простих завдань, наприклад надання довідкової інформації або заповнення форм.

Основна перевага — простота розробки; недолік — відсутність здатності розуміти контекст.

Інтелектуальні (AI-driven) — використовують методи NLP і машинного навчання для розуміння намірів користувача. Такі боти можуть адаптуватися до стилю спілкування, навчатися на попередніх діалогах і підтримувати контекст розмови.

Прикладом є системи на базі Dialogflow, Rasa, ChatGPT API, IBM Watson Assistant.

Гібридні — поєднують сценарійні діалоги з елементами штучного інтелекту. Це дозволяє автоматизувати типові запити, але передавати складні питання людським операторам.

1.3.4 Застосування чат-ботів у клієнтській підтримці

Використання чат-ботів у сфері обслуговування клієнтів охоплює широкий спектр завдань і поступово стає невід'ємною складовою цифрових стратегій підприємств. Такі системи забезпечують автоматизацію стандартних процесів, підвищення швидкості реагування на запити користувачів та створюють позитивний клієнтський досвід.

1. Автоматичні відповіді на типові запити. Одним із найпоширеніших сценаріїв використання чат-ботів є надання швидких відповідей на стандартні питання клієнтів — наприклад, уточнення інформації про товари, послуги, ціни, способи оплати, умови доставки, повернення товарів або графік роботи компанії. Такі запити часто становлять понад 70% від загальної кількості звернень, тому їх автоматизація дозволяє значно зменшити навантаження на операторів служби підтримки. Завдяки цьому фахівці можуть зосередитись на складніших завданнях, що вимагають індивідуального підходу.

2. Підтримка 24/7. На відміну від операторів контакт-центрів, чат-боти функціонують цілодобово, без вихідних і перерв. Це особливо важливо для міжнародних компаній, які обслуговують клієнтів у різних часових поясах. Цілодобова доступність підвищує рівень задоволеності користувачів, адже вони отримують допомогу саме тоді, коли вона їм потрібна, незалежно від робочого графіка підприємства.

3. Персоналізація взаємодії. Сучасні чат-боти здатні аналізувати попередню історію звернень користувача, його уподобання та контекст запиту, формуючи персоналізовані відповіді. Наприклад, система може автоматично пропонувати клієнтові ті продукти чи послуги, якими він раніше цікавився, або враховувати його статус у програмі лояльності. Це створює ефект «розумного асистента» та підвищує довіру до компанії.

4. Збір та аналіз даних. У процесі взаємодії чат-бот накопичує значний обсяг аналітичних даних — частоту звернень, найпоширеніші питання, рівень задоволення користувачів тощо. На основі цих даних менеджмент компанії може

вдосконалювати сервіс, оновлювати базу знань або покращувати процеси взаємодії з клієнтами.

5. Інтеграція з внутрішніми системами підприємства. Бот може підключатися до CRM, ERP або баз даних компанії, що дозволяє автоматично отримувати та оновлювати інформацію — наприклад, перевіряти статус замовлення, формувати заявку або змінювати параметри облікового запису. Така інтеграція робить чат-бота не просто комунікаційним інструментом, а повноцінним елементом бізнес-інфраструктури.

6. Оптимізація витрат на обслуговування. Використання чат-ботів дозволяє підприємствам зменшити витрати на персонал, скоротити час реагування на звернення та підвищити загальну ефективність служби підтримки. Досвід компаній, які впровадили такі системи, показує, що автоматизація навіть 30–40% запитів призводить до помітного економічного ефекту.

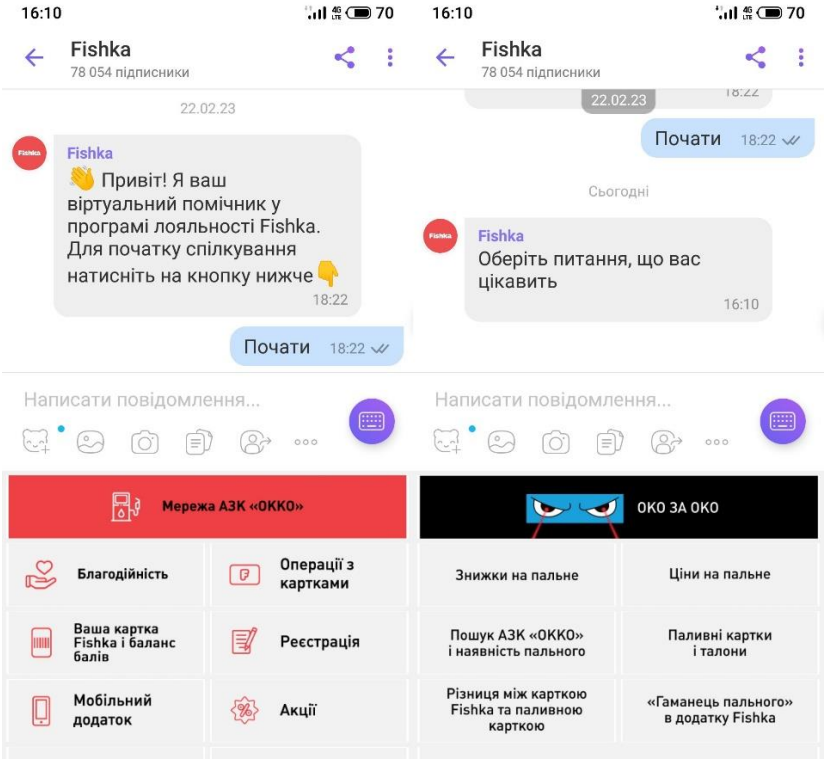


Рисунок 1.6 – Приклад чат боту у Viber

Попередня кваліфікація звернень. Система може класифікувати запити за типом і передавати лише складні випадки оператору.

Збирання статистики та аналітика. Чат-боти фіксують дані про взаємодії, що дозволяє аналізувати поведінку користувачів і вдосконалювати процеси підтримки.

Персоналізація обслуговування. Завдяки збереженню історії взаємодій бот може пропонувати індивідуальні рішення або рекомендації.

Переваги використання чат-ботів

Використання чат-ботів у клієнтській підтримці надає низку вагомих переваг:

1. Скорочення витрат на персонал і інфраструктуру контакт-центру;
2. Підвищення швидкості обслуговування за рахунок миттєвих відповідей;
3. Безперервна робота незалежно від часу доби;
4. Єдині стандарти комунікації та відсутність людського фактору;
5. Можливість масштабування без суттєвих витрат;

Аналітика та звітність, що дозволяє оцінювати ефективність сервісу.

1.3.5 Проблеми та виклики впровадження

Попри численні переваги, використання чат-ботів супроводжується певними труднощами:

обмежене розуміння складних запитів або неформальних висловів користувачів;

1. труднощі у підтримці природності діалогу;
2. необхідність постійного навчання моделей на нових даних;
3. ризики безпеки при роботі з персональними даними;
4. потреба у зручному механізмі передачі діалогу оператору.

Тому ефективне використання чат-ботів вимагає поєднання автоматизації з людським контролем та регулярного вдосконалення алгоритмів обробки мови.

1.4. Технології та платформи для розробки чат-ботів

Розвиток чат-ботів як засобів автоматизації клієнтської підтримки безпосередньо пов’язаний із появою сучасних технологічних рішень і програмних платформ, які спрощують процес їх створення, розгортання та інтеграції з інформаційними системами підприємства. Сьогодні існує велика кількість інструментів, що дозволяють реалізувати чат-боти різної складності — від простих сценарійних систем до інтелектуальних рішень, побудованих на основі штучного інтелекту.

Завдяки розвитку хмарних технологій, відкритих API та фреймворків із відкритим кодом, створення чат-ботів стало доступним навіть для невеликих компаній і окремих розробників. Платформи, такі як Dialogflow, Rasa, Microsoft Bot Framework, Botpress або Telegram Bot API, надають зручні інструменти для побудови діалогових сценаріїв, підключення до баз даних, а також інтеграції з корпоративними системами управління (CRM, ERP, HelpDesk) [6].

Особливої популярності набули рішення, що підтримують обробку природної мови (NLP) і машинне навчання (ML), адже вони дозволяють ботам не просто реагувати на заздалегідь визначені команди, а розуміти контекст запиту, інтерпретувати наміри користувача та адаптувати відповіді. Це суттєво підвищує рівень персоналізації взаємодії та зменшує потребу у втручанні операторів технічної підтримки.

Крім того, завдяки використанню хмарних сервісів (наприклад, AWS Lex, Google Cloud Dialogflow, Azure Bot Service) забезпечується масштабованість, відмовостійкість і безперервність роботи чат-ботів. Такі сервіси дозволяють обробляти тисячі одночасних запитів, що особливо важливо для великих організацій або онлайн-платформ з високим трафіком користувачів.

Важливою тенденцією сучасного етапу розвитку є поєднання чат-ботів із технологіями генеративного штучного інтелекту (Generative AI), що дозволяє створювати гнучкі системи, здатні вести природні діалоги, підтримувати контекст протягом тривалого спілкування та формувати динамічні відповіді. Моделі на основі архітектури Transformer (зокрема GPT, BERT, T5) стали основою для створення високорівневих мовних агентів, які вже використовуються у сфері клієнтського сервісу, освіти, медицини та електронної комерції.

Отже, сучасні технологічні рішення відкрили новий етап розвитку чат-ботів — від простих автоматизованих асистентів до інтелектуальних систем взаємодії. Такий підхід дозволяє підприємствам не лише знижувати витрати на підтримку користувачів, але й формувати позитивний досвід взаємодії з клієнтами, підвищуючи лояльність та ефективність комунікації.

1.4.1 Архітектура системи чат-бота

Типова архітектура сучасного чат-бота включає такі основні компоненти (рис. 1.1):

Інтерфейс взаємодії з користувачем (User Interface Layer) — забезпечує прийом і відображення повідомлень у вибраному середовищі: вебчат, Telegram, Facebook Messenger, WhatsApp, Slack, Viber тощо.

Серверна логіка (Backend Layer) — обробляє запити, керує станом діалогу, викликає алгоритми аналізу тексту, приймає рішення та формує відповіді.

Модуль NLP / NLU (Natural Language Processing / Understanding) — відповідає за обробку природної мови, виділення сутностей, визначення намірів користувача (intent recognition) і формування структурованих даних.

База знань (Knowledge Base) — містить інформацію, сценарії діалогів, шаблони відповідей, FAQ, а також можливі інтеграції з базами даних чи CRM.

Зовнішні сервіси та API (Integration Layer) — дозволяють чат-боту отримувати або надсилати дані в інші системи (наприклад, перевірити статус замовлення, створити заявку, зберегти відгук тощо).

Подібна архітектура забезпечує модульність, масштабованість і можливість подальшого вдосконалення системи без необхідності повного її перепроектування.

1.4.2 Класифікація технологій для створення чат-ботів

Технології розробки чат-ботів можна поділити на кілька основних категорій:

Платформи з графічними інтерфейсами (No-code / Low-code solutions) Такі сервіси дозволяють створювати бота без глибоких знань програмування, використовуючи візуальні редактори сценаріїв. Приклади:

Dialogflow (Google Cloud) — потужна NLP-платформа, яка дозволяє будувати діалоги, визначати наміри користувачів і інтегрувати бота у різні канали комунікації.

Microsoft Bot Framework + Azure Bot Service — корпоративне рішення для створення ботів, орієнтованих на інтеграцію з екосистемою Microsoft (Teams, Dynamics 365, Azure Cognitive Services).

ManyChat, Chatfuel — інструменти для створення маркетингових ботів у Facebook Messenger, Instagram, WhatsApp, орієнтовані на бізнес-користувачів.

Фреймворки з відкритим кодом (Open-source frameworks) Вони забезпечують більшу гнучкість, можливість налаштування логіки обробки запитів і інтеграцію з AI-модулями:

Rasa — одна з найпопулярніших open-source платформ для створення AI-ботів. Складається з двох основних компонентів: *Rasa NLU* (аналіз природної мови) і *Rasa Core* (керування діалогом).

Botpress — фреймворк із модульною архітектурою, що підтримує сценарії на JavaScript і має вбудований графічний інтерфейс.

ChatterBot (Python) — бібліотека для побудови простих текстових ботів із функціями самонавчання.

API та бібліотеки для розробки (Developer-oriented tools) Ці інструменти орієнтовані на розробників, які створюють власні рішення із застосуванням мов програмування (зокрема Python, JavaScript, C#):

Telegram Bot API — офіційний інтерфейс для створення ботів у Telegram, який надає розвинені можливості інтерактивної взаємодії, зокрема клавіатури, inline-запити, кнопки та мультимедіа.

OpenAI API (ChatGPT, GPT-4, GPT-5) — дозволяє реалізувати генеративну модель спілкування, що розуміє контекст, формує природні відповіді й адаптується до поведінки користувача.

Python-бібліотеки:

aiogram — асинхронна бібліотека для створення Telegram-ботів;

pyTelegramBotAPI — проста у використанні бібліотека для інтеграції з Telegram;

NLTK, spaCy, transformers — для обробки природної мови та реалізації NLP-модулів.

Хмарні сервіси штучного інтелекту (AI-powered services) Вони надають готові інструменти для NLP, розпізнавання мови, перекладу та аналізу настроїв:

Google Cloud Natural Language API;

IBM Watson Assistant;

Amazon Lex (AWS);

Azure Cognitive Services. Такі сервіси дозволяють розробникам інтегрувати складні AI-функції без необхідності навчати власні моделі.

1.4.3 Мови програмування та технології інтеграції

Для реалізації чат-ботів найчастіше використовуються такі технології:

Python — найбільш популярна мова для створення інтелектуальних ботів завдяки розвиненій екосистемі бібліотек AI та NLP.

JavaScript / TypeScript (Node.js) — часто застосовується для створення ботів у вебсередовищах або інтеграції з мікросервісами.

C# — використовується в корпоративних рішеннях, зокрема у Microsoft Bot Framework.

Webhooks і REST API — забезпечують обмін даними між ботом і зовнішніми системами, такими як CRM, ERP або бази даних.

Інтеграція може здійснюватися як через прямі HTTP-запити, так і через посередницькі сервіси (наприклад, Zapier або n8n).

```
const Http = new XMLHttpRequest();
const url='https://jsonplaceholder.typicode.com/posts';
Http.open("GET", url);
Http.send();

Http.onreadystatechange=(e)=>{
    console.log(Http.responseText)
}
```

Рисунок 1.7 – Приклад HTTP запиту у JavaScript

Вимоги до сучасних платформ для створення чат-ботів

Сучасні інструменти для побудови ботів повинні забезпечувати:

1. гнучкість та масштабованість — можливість обробляти велику кількість запитів одночасно;
2. підтримку багатомовності;
3. можливість навчання на історичних даних;
4. захист персональних даних користувачів;
5. інтеграцію з різними каналами комунікації;
6. аналітику та моніторинг ефективності роботи.

Виконання цих вимог дозволяє створювати надійні та адаптивні рішення для клієнтської підтримки в різних сферах діяльності.

Висновок до розділу 1

У першому розділі роботи було проведено теоретичний аналіз проблеми автоматизації клієнтської підтримки та визначено роль чат-ботів як ключового інструменту її реалізації в сучасних інформаційних системах. Розглянуто сутність

клієнтської підтримки, її значення для підвищення ефективності взаємодії між організацією та користувачем, а також основні недоліки традиційних підходів, що обумовлюють необхідність упровадження автоматизованих рішень.

Проаналізовано еволюцію технологій чат-ботів — від простих сценарійних систем до інтелектуальних агентів, які використовують методи машинного навчання та обробки природної мови (NLP). Особливу увагу приділено принципам побудови таких систем, типам їх архітектури та ключовим компонентам, що забезпечують ефективність обробки користувацьких запитів.

Визначено основні підходи до класифікації чат-ботів за функціональністю, способом взаємодії та ступенем інтелектуальності. Проведено огляд сучасних технологій, фреймворків і платформ для створення чат-ботів, зокрема Dialogflow, Rasa, Botpress, Microsoft Bot Framework, а також інструментів для інтеграції з популярними комунікаційними сервісами, такими як Telegram, Slack чи Facebook Messenger.

Окремо відзначено роль мов програмування (Python, JavaScript) та бібліотек для реалізації NLP і AI-функцій, що дозволяють розробляти адаптивні, гнучкі та масштабовані рішення.

Таким чином, у результаті проведеного аналізу встановлено, що використання чат-ботів є перспективним напрямом у сфері автоматизації клієнтської підтримки. Інтеграція таких систем у бізнес-процеси дає змогу підвищити швидкість реагування, зменшити навантаження на персонал, покращити якість обслуговування та забезпечити безперервну доступність сервісів.

Отримані теоретичні результати створюють основу для подальшого проєктування та реалізації програмного прототипу чат-бота, що буде розглянуто в наступному розділі.

2. АНАЛІЗ ТА РОЗРОБКА МОДЕЛІ СИСТЕМИ АВТОМАТИЗАЦІЇ КЛІЄНТСЬКОЇ ПІДТРИМКИ

2.1. Вибір інструментів та середовища розробки

Розробка системи автоматизації клієнтської підтримки на основі чат-бота передбачає ретельний вибір технологій, які забезпечать ефективність, гнучкість та надійність майбутнього рішення. Вибір інструментів безпосередньо впливає на продуктивність системи, простоту інтеграції з іншими сервісами, можливість масштабування та подальшого супроводу.

Одним із ключових етапів є визначення мови програмування, оскільки саме вона формує основу логіки роботи чат-бота, визначає доступність бібліотек для обробки природної мови (NLP), машинного навчання (ML) та інтеграції з API месенджерів. Для реалізації даного проєкту було розглянуто кілька популярних мов: Python, JavaScript (Node.js), Java, C# та PHP.

Таблиця 2.1 – Порівняння мов програмування

Мова програмування	Переваги	Недоліки	Придатність для чат-ботів
Python	Простий синтаксис, наявність потужних бібліотек для NLP, AI та ML (NLTK, spaCy, TensorFlow, scikit-learn), активна спільнота, зручна робота з API.	Нижча швидкість виконання порівняно з компільованими мовами.	**Висока** — оптимальний вибір для інтелектуальних ботів.
JavaScript (Node.js)	Асинхронність, висока швидкість у мережевих застосунках, велика кількість бібліотек для роботи з API.	Обмежена підтримка AI/NLP, складніша реалізація ML-завдань.	**Висока** — ефективна для інтерактивних і швидких ботів.
Java	Висока продуктивність, стабільність, багатопоточність, кросплатформеність.	Складніша розробка, громіздкий код, менше бібліотек для NLP.	**Середня** — доцільна для великих корпоративних систем.

Продовження таблиці 2.1

C#	Добра інтеграція з Microsoft Bot Framework, надійність.	Обмежена кросплатформеність, залежність від екосистеми Microsoft.	**Середня** — ефективна в корпоративному середовищі Microsoft.
PHP	Простота інтеграції з вебсерверами, популярність у веброзробці.	Низька швидкість, слабка підтримка асинхронності, обмежені можливості AI/NLP.	**Низька** — не підходить для інтелектуальних ботів.

Як видно з порівняння, Python має найкраще співвідношення між простотою використання, кількістю спеціалізованих бібліотек і можливістю інтеграції з технологіями штучного інтелекту. Це робить його найбільш придатною мовою для створення чат-ботів, орієнтованих на аналіз природної мови та навчання на основі даних користувачів[11].

Python підтримує широкий спектр бібліотек і фреймворків, таких як:

aiogram — асинхронний фреймворк для створення Telegram-ботів;

NLTK та spaCy — для аналізу й обробки природної мови;

scikit-learn, TensorFlow та PyTorch — для реалізації алгоритмів машинного навчання;

SQLite — для локального збереження даних і журналів звернень користувачів.

```python
import asyncio
import logging
from aiogram import Bot, Dispatcher, types
from aiogram.filters.command import Command

# Включаем логирование, чтобы не пропустить важные сообщения
logging.basicConfig(level=logging.INFO)
# Объект бота
bot = Bot(token="12345678:AaBbCcDdEeFfGgHh")
# Диспетчер
dp = Dispatcher()

# Хэндлер на команду /start
@dp.message(Command("start"))
async def cmd_start(message: types.Message):
    await message.answer("Hello!")

# Запуск процесса поллинга новых апдейтов
async def main():
    await dp.start_polling(bot)

if __name__ == "__main__":
    asyncio.run(main())
```

Рисунок 2.1 – Приклад використання aiogram

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged[0:6]
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
('Thursday', 'NNP'), ('morning', 'NN')]
```

Рисунок 2.2 – Приклад використання NLTK

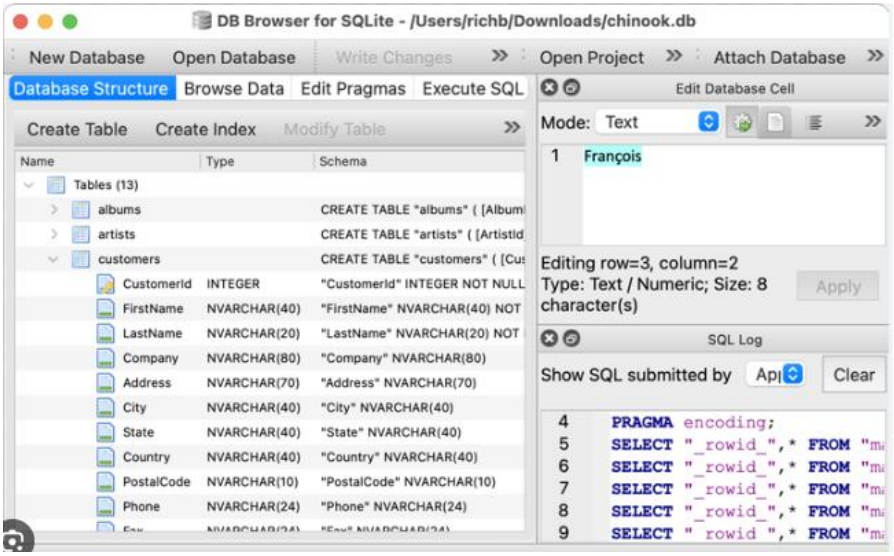


Рисунок 2.3 – Інтерфейс SQLite

У межах даної роботи для взаємодії з користувачем обрано Telegram Bot API, який забезпечує просту інтеграцію, стабільну роботу та підтримку сучасних функцій (inline-кнопки, клавіатури, обробку мультимедійних повідомлень тощо).

Розробку системи здійснено у середовищі Visual Studio Code, що забезпечує зручність написання коду, підтримку плагінів для Python, інтеграцію з системою контролю версій Git і можливість швидкого розгортання застосунку у хмарних середовищах (Heroku, Render тощо).

Таким чином, вибір мови Python та відповідних технологічних інструментів є оптимальним рішенням, оскільки забезпечує:

1. просту реалізацію інтелектуальної логіки чат-бота;
2. ефективну роботу з текстовими даними та користувацькими запитами;
3. масштабованість і гнучкість системи;
4. мінімальні витрати часу на розробку та супровід.

2) Безкодовий метод

У наш час існує велика потребність у автоматизації клієнської підтриимки, проте не у всіх є можливість та навички для їх створення класичним методом. У таких випадках на допоммогу приходять онлайн сервіси, що допомагають створити власного чат бота з потрібними саме тобі функціями без пробел та потрібності начання мовам програмування.

Існує багато сервісів що задовольняють наші потреби. Найпопулярнішими варіантами є:

ManyChat — один з найвідоміших конструкторів з візуальним редактором, підтримує автоматизацію, розсилки, інтеграції з CRM. Має безкоштовний план для базових функцій.

Chatfuel — зручний drag-and-drop інтерфейс, добре підходить для маркетингових ботів, є AI-функції для обробки запитів. Популярний серед бізнесу.

BotHelp — український сервіс з російськомовним інтерфейсом, спеціалізується на ботах для підтримки клієнтів та продажів, має готові шаблони.

SendPulse — комплексна платформа для маркетингу, дозволяє створювати ботів без коду з підтримкою розсилок, чат-ботів та автоворонок.

Aimylogic — платформа з візуальним конструктором діалогів, підтримує AI для розпізнавання намірів користувачів.

BotKits — простий конструктор з готовими блоками для швидкого запуску ботів.

Landbot — сучасний інтерфейс, добре підходить для створення конверсійних воронок через чат-ботів.

Кожна з цих платформ може задовольнити наші потреби тому зробимо більш детальний аналіз преваг та недоліків кожної з них.

Таблиця 2.2 – Порівняння сервісів безкодового створення чат-ботів

Критерій	BotHelp	SendPulse	ManyChat	Chatfuel
Вартість (базовий план)	Від $15/міс	Безкоштовно до 500 контактів	Від $15/міс	Від $15/міс
Мова інтерфейсу	Українська/Російська	Українська/Російська	Англійська	Англійська
Прийом заявок	✓ Спеціалізовані форми	✓ Базові форми	✓ Базові форми	✓ Базові форми
CRM-інтеграція	✓ Вбудована CRM	✓ Інтеграції	✓ Інтеграції	✓ Обмежені
Відстеження статусу замовлень	✓ Нативна підтримка	~ Через налаштування	~ Через налаштування	✗

Продовження таблиці 2.2

Запис на прийом	✓ Календар	✓ Через інтеграції	✓ Через інтеграції	✗
Автоматичні нагадування	✓	✓	✓	✓
Складність налаштування	Низька	Середня	Середня	Середня
Спеціалізація	Сервісний бізнес	Маркетинг	Маркетинг	Маркетинг
Технічна підтримка	Українською	Українською	Англійською	Англійською
Готові шаблони для сервісу	✓ Багато	~ Обмежено	~ Обмежено	✗

Обґрунтування вибору BotHelp

1. Спеціалізація на сервісних бізнесах

BotHelp створювався спеціально для компаній, що надають послуги, включаючи ремонтні сервіси. Платформа містить готові рішення для:

1. Прийому та обробки заявок на ремонт
2. Відстеження статусу замовлень
3. Управління чергою клієнтів
4. Комунікації з клієнтами на всіх етапах обслуговування

На відміну від ManyChat та Chatfuel, які орієнтовані на маркетинг та продажі, BotHelp зосереджений на операційних процесах сервісного бізнесу.

2. Вбудована CRM-система

Платформа має власну CRM, що дозволяє:

1. Зберігати історію звернень кожного клієнта
2. Відстежувати етапи виконання замовлення (прийнято → діагностика → ремонт → готово)
3. Управляти базою даних техніки та типових несправностей
4. Аналізувати ефективність роботи сервісу

Інші платформи потребують додаткових інтеграцій з зовнішніми CRM-системами, що ускладнює налаштування та збільшує вартість.

3. Мовна локалізація

Повна підтримка української та російської мов критично важлива для:

1. Швидкого налаштування без мовного бар'єру
2. Комфортної роботи персоналу сервісу
3. Отримання технічної підтримки рідною мовою
4. Природної комунікації з україномовними клієнтами

4. Спеціалізовані функції для ремонтних сервісів

BotHelp пропонує унікальні можливості:

Форми прийому заявок з полями для моделі техніки, серійного номера, опису проблеми

Завантаження фото пошкоджень для попередньої діагностики

Система статусів з автоматичним оновленням клієнта

Календар записів для планування візитів клієнтів

Шаблони повідомлень для типових ситуацій (техніка прийнята, діагностика завершена, ремонт готовий)

5. Економічна ефективність

Порівняно з конкурентами:

1. SendPulse має безкоштовний план, але обмежений функціонал для сервісного бізнесу
2. ManyChat та Chatfuel потребують додаткових витрат на інтеграції
3. BotHelp за схожу ціну надає комплексне рішення без необхідності підключення сторонніх сервісів

6. Простота впровадження

Низький поріг входу завдяки:

1. Інтуїтивному візуальному конструктору
2. Готовим шаблонам для ремонтних сервісів
3. Детальній документації українською мовою
4. Можливості запуску базового бота за 1-2 години

Для впровадження чат-бота в сервісі з ремонту електротехніки платформа BotHelp є оптимальним вибором завдяки спеціалізації на сервісному бізнесі, наявності вбудованої CRM, повній локалізації та набору функцій, спеціально розроблених для управління процесом ремонту. Альтернативні платформи (SendPulse, ManyChat, Chatfuel) орієнтовані переважно на маркетингові завдання та потребують значно більших зусиль для адаптації під специфіку ремонтного сервісу.

2.2. Архітектура та структура системи клієнтської підтримки на основі чат-бота

Архітектура системи автоматизації клієнтської підтримки визначає взаємодію між усіма її компонентами та забезпечує узгоджене виконання основних функцій — отримання, аналізу та обробки користувацьких запитів. Вона формує логічну структуру побудови системи, у межах якої кожен модуль має чітко визначені обов’язки, точки інтеграції та канали обміну даними[15].

При розробці системи було застосовано модульний підхід, що дає змогу розділити програму на окремі функціональні частини, кожна з яких відповідає за певний етап обробки інформації. Така структура спрощує процес оновлення, тестування та масштабування системи, оскільки зміни в одному компоненті не потребують переробки всієї архітектури. Завдяки цьому можна гнучко адаптувати

рішення до потреб різних підприємств або сервісних служб — від невеликих локальних компаній до великих організацій з розгалуженою мережею користувачів.

Ключовим завданням архітектури є забезпечення високої надійності та ефективної взаємодії між компонентами. Усі модулі системи працюють у тісній інтеграції, обмінюючись даними через стандартизовані інтерфейси. Це дозволяє підтримувати узгодженість інформаційних потоків і зменшити ризик виникнення помилок під час передачі запитів або формування відповідей.

Особливу увагу приділено масштабованості системи — можливості збільшення кількості користувачів і обсягу запитів без зниження продуктивності. Це досягається за рахунок використання асинхронної обробки повідомлень, оптимізації запитів до бази даних та ефективного управління ресурсами.

Крім того, архітектура передбачає гнучкість у розгортанні: систему можна реалізувати як локально, так і в хмарному середовищі, що забезпечує мобільність, швидке оновлення та зручність обслуговування.

Важливою особливістю є підтримка розширюваності, тобто можливості підключення додаткових модулів, таких як аналітичні панелі, інтеграція зі сторонніми CRM-системами або модулі машинного навчання для самонавчання чат-бота. Це дозволяє поступово підвищувати рівень автоматизації без повної перебудови існуючої системи.

Таким чином, обрана архітектура поєднує простоту реалізації з високою функціональністю та забезпечує основу для подальшого розвитку інтелектуальних сервісів клієнтської підтримки.

2.2.1. Загальна структура системи

Загальна структура системи автоматизації клієнтської підтримки базується на багаторівневій архітектурі, що включає чотири основні рівні: інтерфейс взаємодії з користувачем, рівень логіки обробки запитів, рівень даних та аналітичний рівень. Такий підхід забезпечує логічну ізоляцію функцій, що підвищує стабільність і гнучкість системи.

Інтерфейс взаємодії з користувачем. На цьому рівні відбувається безпосередня комунікація користувача із системою. Основним компонентом є чат-бот, реалізований у середовищі Telegram із використанням Telegram Bot API. Через цей інтерфейс користувач надсилає текстові повідомлення, запити чи команди, які далі передаються для аналізу та обробки. Завдяки простоті інтеграції й широкій аудиторії користувачів Telegram став оптимальним вибором для реалізації системи.

Рівень логіки обробки запитів. Цей рівень виконує ключову функцію – інтерпретацію отриманих запитів і формування відповідей. Тут реалізовано механізми розпізнавання інтенцій користувача, обробки природної мови (через бібліотеки Python, зокрема NLTK, spaCy або transformers) і прийняття рішень

згідно з визначеним сценарієм. Логіка побудована таким чином, щоб система могла адаптуватися до контексту діалогу й надавати релевантні відповіді.

Рівень даних. На цьому етапі відбувається зберігання, пошук і оновлення інформації, необхідної для роботи чат-бота. Використовується база даних, де зберігаються шаблони діалогів, історія спілкувань, дані користувачів і службова інформація про стан системи. Для цього може бути застосована SQLite або PostgreSQL, залежно від масштабів проєкту. Забезпечується можливість швидкого доступу до даних і підтримка транзакцій для збереження цілісності інформації.

Аналітичний рівень. Призначений для збору статистики та оцінки ефективності роботи чат-бота. Він здійснює моніторинг частоти звернень, часу реакції, рівня задоволеності користувачів і виявляє найбільш типові запити. На основі зібраних даних можлива подальша оптимізація сценаріїв і вдосконалення моделей машинного навчання.

Усі рівні взаємодіють між собою через API-з'єднання або внутрішні функціональні виклики. Така структура дозволяє гнучко модифікувати будь-який компонент без потреби зміни всієї системи. Наприклад, заміна Telegram на іншу платформу (Viber, Discord, веб-чат) потребуватиме лише коригування інтерфейсного рівня, не зачіпаючи основну логіку.

Таким чином, розроблена архітектура поєднує модульність, масштабованість і аналітичну складову, що забезпечує ефективну роботу системи клієнтської підтримки та створює передумови для її подальшого розвитку.

Процес взаємодії користувача з системою відбувається за наступним алгоритмом:

1. Користувач надсилає повідомлення або запит у Telegram-бот.
2. Модуль комунікації отримує запит через Telegram Bot API та передає його в модуль NLP.
3. Модуль обробки природної мови аналізує текст, визначає контекст і намір користувача.
4. Модуль логіки приймає рішення щодо подальших дій — надання відповіді, виконання операції або звернення до бази даних.
5. Відповідь формується та надсилається користувачу через Telegram.
6. Усі запити зберігаються в базі даних для подальшого аналізу та вдосконалення роботи чат-бота.

Під час розробки системи дотримано таких принципів:

- Модульність — кожен компонент виконує окрему функцію, що полегшує оновлення та тестування системи.

- Масштабованість — можливість додавання нових модулів або розширення існуючих без значних змін у загальній структурі.

- Відмовостійкість — у разі збою одного модуля інші можуть продовжувати роботу.

- Гнучкість — адаптація системи до різних типів організацій і сценаріїв взаємодії з користувачем.

- Безперервність обслуговування — робота чат-бота 24/7 без потреби у постійній присутності оператора.

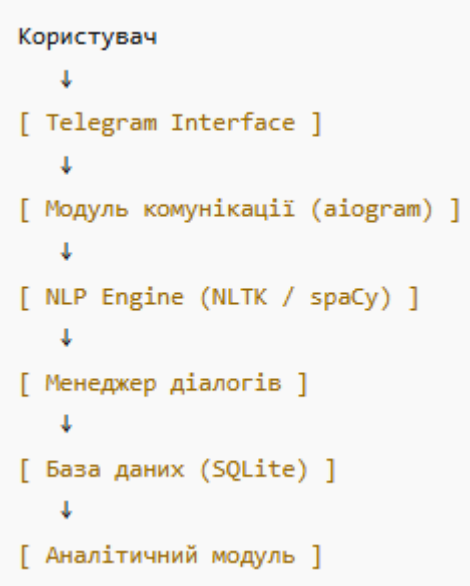


Рисунок 2.4 – Візуалізація структури системи

2.3. Опис алгоритмів обробки запитів користувача

Алгоритм роботи чат-бота у системі автоматизації клієнтської підтримки побудований на основі послідовного аналізу вхідного повідомлення, визначення його змісту (інтенції) та формування відповідної відповіді. Для реалізації цього процесу використано комбінацію методів обробки природної мови (NLP), класифікації текстів та сценарного управління діалогом. Такий підхід дозволяє створити систему, здатну ефективно обробляти запити користувачів у реальному часі, забезпечуючи логічну послідовність діалогу та збереження контексту розмови.

Основна мета алгоритму — забезпечити коректне розуміння запиту користувача незалежно від формулювання, граматичних особливостей чи синонімічних варіацій. Це досягається завдяки попередній обробці тексту, що включає токенізацію, лематизацію, видалення стоп-слів, а також визначення семантичних зв’язків між словами. Далі застосовується модель машинного навчання або нейронна мережа, навчена на наборі прикладів діалогів, для класифікації повідомлення за певними інтенціями — типами запитів, які відповідають конкретним сценаріям взаємодії.

Кожна інтенція пов’язана з певним контекстом і набором можливих дій, які чат-бот може виконати: надати інформацію, перейти до наступного етапу діалогу, або звернутися до зовнішнього джерела даних. Для підвищення гнучкості алгоритм використовує механізм сценарного управління діалогом (dialog

management), який дозволяє підтримувати природний перебіг розмови, враховуючи попередні повідомлення користувача.

Крім того, у системі передбачено механізм динамічного оновлення бази знань, що дає змогу боту адаптуватися до нових типів запитів без повного перенавчання моделі. У поєднанні з аналітичним модулем, який відстежує помилки класифікації та рівень задоволеності користувачів, це створює основу для постійного вдосконалення якості взаємодії.

Таким чином, алгоритм обробки запитів поєднує традиційні правила обробки тексту з інтелектуальними моделями машинного навчання, формуючи гібридний підхід, що забезпечує точність, швидкість і адаптивність у системах клієнтської підтримки.

2.3.1. Загальна схема обробки запиту

Процес взаємодії користувача з чат-ботом можна подати у вигляді таких етапів:

Отримання повідомлення. Користувач надсилає повідомлення до чат-бота через Telegram. Повідомлення передається через Telegram API, після чого бот зчитує текст і формує вхідний об’єкт запиту.

Попередня обробка тексту. Текст очищується від пунктуації, спеціальних символів, переводиться у нижній регістр, а також здійснюється токенізація (розбиття на слова) і лематизація (зведення до початкової форми). Для цього застосовуються бібліотеки NLTK або spaCy.

Визначення інтенції (намір користувача). На цьому етапі виконується аналіз змісту повідомлення. За допомогою моделей машинного навчання або правил логічного зіставлення система визначає, до якої категорії належить запит (наприклад: “Отримати довідку”, “Звернутися до оператора”, “Інформація про оплату”, “Графік роботи” тощо). Для цього може використовуватись:

Модель класифікації на основі навчання з учителем (Logistic Regression, SVM, Naive Bayes);

або нейронна модель (наприклад, BERT або DistilBERT), якщо система має достатню кількість навчальних даних.

Пошук відповіді. Після визначення інтенції система звертається до бази знань або шаблонів діалогів. Якщо запит належить до відомої категорії, бот формує відповідь із заздалегідь підготовлених даних. У разі складного або невизначеного запиту система може звернутися до LLM-моделі (наприклад, GPT API) або переадресувати запит оператору.

Формування відповіді. На основі знайденої інформації система створює структуровану відповідь, адаптовану до формату платформи (наприклад, текст, кнопки з варіантами дій, посилання, зображення).

Надсилання результату користувачу. Відповідь надсилається назад через Telegram API. В історії діалогу зберігається інформація про запит, відповідь і часову мітку для подальшого аналізу статистики.

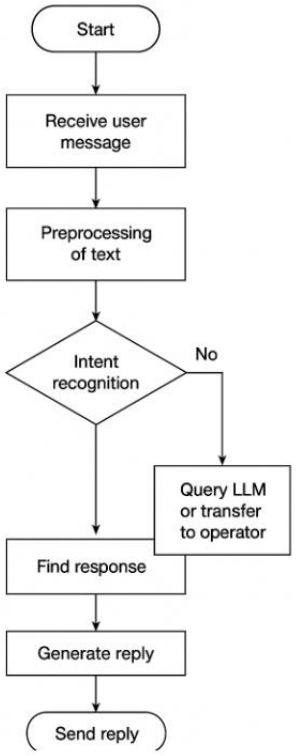


Рисунок 2.5 – Алгоритм роботи

2.3.2. Опис загального алгоритму роботи

Умовно алгоритм роботи чат-бота системи автоматизації клієнтської підтримки можна представити у вигляді послідовності взаємопов'язаних етапів, які забезпечують повний цикл обробки користувацького запиту — від моменту його надходження до формування відповіді.

Start (Початок роботи) Система переходить у стан очікування нового запиту від користувача через інтегрований канал — вебінтерфейс, месенджер або іншу платформу[12].

Отримання вхідного повідомлення користувача Після надходження повідомлення бот реєструє його у базі даних із зазначенням часу, джерела запиту та ідентифікатора користувача. Цей етап ініціює подальшу обробку тексту.

Попередня обробка тексту (Preprocessing) Вхідний текст очищується від зайвих символів, пунктуації, HTML-тегів, емоджі та стоп-слів. Далі проводиться токенізація, лематизація та нормалізація — процеси, що перетворюють текст у зручний формат для машинного аналізу.

Визначення інтенції за допомогою NLP-моделі На цьому етапі оброблений текст подається на вхід моделі класифікації, яка визначає інтенцію (намір) користувача — наприклад, «отримати інформацію», «змінити налаштування» або

«залишити скаргу». Для цього використовуються алгоритми машинного навчання або попередньо навчена модель NLP.

Перевірка розпізнавання інтенції Якщо інтенцію розпізнано з високою ймовірністю, система переходить до етапу пошуку відповіді у відповідній базі знань. Якщо ж модель не впевнена у результаті, чат-бот може виконати один із двох сценаріїв:

Звернення до LLM (Large Language Model) — генерація відповіді на основі великої мовної моделі, наприклад GPT;

Передача запиту оператору, якщо потрібно людське втручання.

Пошук релевантної відповіді у базі знань На цьому етапі здійснюється звернення до бази знань (Knowledge Base), яка містить попередньо підготовлені відповіді, шаблони повідомлень або інструкції. Пошук здійснюється за ключовими словами, семантичною близькістю або категорією інтенції.

Формування відповіді Вибрана відповідь проходить етап форматування: додаються необхідні динамічні дані (наприклад, ім’я користувача, посилання, контактна інформація), після чого формується повідомлення у зручному для відображення вигляді.

Надсилання повідомлення користувачу Готова відповідь відправляється через відповідний канал комунікації — месенджер, вебчат чи мобільний додаток. У разі використання мультимедійних елементів (зображень, кнопок, посилань) система підлаштовує формат під можливості платформи.

Збереження даних у базі Всі дані про діалог (текст запиту, відповідь, час реакції, ідентифікатор оператора або модуля, який сформував відповідь) записуються у журнал звернень. Ця інформація надалі використовується для аналітики, навчання моделі та підвищення якості обслуговування.

End (Завершення) Після відправлення відповіді чат-бот повертається у стан очікування наступного запиту, зберігаючи при цьому контекст поточного діалогу для можливого продовження розмови.

Таким чином, описаний алгоритм забезпечує безперервну та адаптивну взаємодію між користувачем і системою підтримки. Він поєднує в собі автоматичну обробку запитів, інтелектуальні технології NLP та гнучку архітектуру інтеграції, що робить систему придатною як для малих компаній, так і для великих корпоративних структур.

Для підвищення точності розпізнавання запитів у системі передбачено:

Використання словників синонімів та контекстних шаблонів, що дозволяє враховувати різні варіанти формулювань;

Механізм контекстної пам’яті, який зберігає кілька попередніх повідомлень користувача, забезпечуючи більш природну взаємодію у діалозі;

Можливість самонавчання системи на основі накопичених історій діалогів і зворотного зв’язку користувачів.

Таким чином, алгоритм роботи чат-бота поєднує елементи класичного оброблення тексту з інтелектуальними методами машинного навчання, що дозволяє досягти високої ефективності при автоматизації клієнтської підтримки. Гнучка архітектура алгоритму забезпечує можливість масштабування системи та інтеграції нових модулів без зміни основної логіки роботи.

2.3.3. Особливості реалізації

Для підвищення точності розпізнавання запитів користувачів у системі реалізовано низку інтелектуальних механізмів, спрямованих на вдосконалення процесу обробки природної мови.

По-перше, застосовано словники синонімів, контекстних шаблонів і тематичних категорій, які дозволяють системі розуміти різні формулювання одного й того ж запиту. Наприклад, запит *«як змінити пароль»* може бути розпізнаний навіть у формі *«забув пароль», «оновити вхідні дані»* або *«проблема з входом»*. Це забезпечується попередньою обробкою тексту з використанням лематизації, токенізації та аналізу частин мови.

По-друге, у чат-боті реалізовано механізм контекстної пам’яті, що зберігає кілька попередніх повідомлень користувача. Завдяки цьому бот може підтримувати логіку діалогу й адекватно реагувати на уточнюючі повідомлення. Наприклад, якщо користувач після питання *«як оплатити рахунок?»* напише *«а якщо карткою?»*, система зрозуміє, що уточнення стосується попереднього запиту. Такий підхід значно підвищує природність взаємодії, наближаючи спілкування до живої розмови.

По-третє, реалізовано механізм самонавчання системи, який базується на аналізі накопичених історій діалогів і відгуків користувачів. Після кожної сесії спілкування система може автоматично оновлювати свою базу знань, удосконалюючи класифікатор інтенцій. Це дозволяє поступово підвищувати точність визначення запитів і зменшувати кількість помилкових відповідей без необхідності ручного втручання розробника.

Додатково передбачено систему зворотного зв’язку, за допомогою якої користувач може оцінити відповідь чат-бота як коректну або некоректну. Такі оцінки надалі враховуються при навчанні моделі, що сприяє адаптації чат-бота до реальних потреб аудиторії.

Таким чином, алгоритм роботи чат-бота поєднує класичні методи оброблення тексту (NLP) з інтелектуальними підходами машинного навчання, що забезпечує високу ефективність автоматизації клієнтської підтримки. Його гнучка архітектура дозволяє легко масштабувати систему, додавати нові функціональні модулі, підключати зовнішні джерела даних або інтегрувати рішення з корпоративними CRM-системами без необхідності зміни базової логіки роботи.

2.4. Особливості інтеграції чат-бота з базою даних та зовнішніми сервісами

Інтеграція чат-бота з базами даних та зовнішніми сервісами є одним із ключових аспектів побудови ефективної системи автоматизації клієнтської підтримки. Саме вона забезпечує можливість доступу до актуальної інформації, персоналізації відповідей, а також взаємодію з внутрішніми бізнес-процесами підприємства.

1. Роль бази даних у роботі чат-бота

База даних (БД) виступає центральним елементом інформаційної структури системи. Вона зберігає всі необхідні відомості для функціонування чат-бота, зокрема:

історію діалогів з користувачами (тексти повідомлень, дати, результати обробки);

дані користувачів (ідентифікатор, контактні дані, статус клієнта, попередні звернення);

базу знань (запитання-відповіді, шаблони діалогів, категорії запитів);

журнали подій (помилки, статистику використання, показники ефективності);

словники ключових слів, синонімів та моделей NLP, що використовуються для аналізу тексту.

Для реалізації було використано реляційну базу даних, побудовану на SQLite / PostgreSQL, яка забезпечує зручність інтеграції з Python-застосунками через ORM-бібліотеки (наприклад, SQLAlchemy).

Таблиця 2.3 – Типова структура

Таблиця	Призначення	Основні поля
`users`	Зберігає дані користувачів	`user_id, name, contact, last_interaction`
`dialogs`	Містить історію діалогів	
`knowledge_base`	База знань (запитання-відповідь)	`intent, question_pattern, answer_text, tags`
`logs`	Технічна інформація про роботу системи	`event_id, event_type, details, created_at`

3. Інтеграція чат-бота із зовнішніми сервісами

Сучасні чат-боти рідко функціонують ізольовано. Для забезпечення повноцінної роботи необхідна інтеграція із зовнішніми інформаційними

системами, API та платформами комунікацій. У межах даної розробки передбачено такі напрями взаємодії:

Інтеграція з месенджерами (Telegram API, Facebook Messenger API, Viber API) — забезпечує приймання та відправлення повідомлень, обробку кнопок, медіа та реакцій користувачів.

Взаємодія з CRM-системами (Customer Relationship Management) — передача даних про звернення клієнтів у систему управління відносинами, що дозволяє автоматично створювати заявки, завдання або тікети.

Підключення до аналітичних сервісів — передача статистики роботи чат-бота (кількість звернень, середній час відповіді, рівень задоволення користувачів) до систем звітності або візуалізації (наприклад, Google Data Studio, Power BI).

Інтеграція із сервісами машинного навчання та NLP (Dialogflow, OpenAI API, Hugging Face Transformers) — для аналізу природної мови, визначення інтенцій та генерації відповідей.

4. Безпека та захист даних

Під час інтеграції з БД та зовнішніми сервісами важливо дотримуватись принципів інформаційної безпеки:

1. усі запити виконуються через захищені з’єднання (HTTPS, SSL/TLS);
2. доступ до API здійснюється за токенами або ключами авторизації;
3. дані користувачів зберігаються у зашифрованому вигляді;
4. реалізовано механізми контролю доступу до різних рівнів даних.

5. Узагальнення

Таким чином, інтеграція чат-бота з базою даних і зовнішніми сервісами формує інформаційну екосистему підтримки клієнтів, у якій усі компоненти взаємодіють у режимі реального часу.

Завдяки цьому система:

1. оперативно отримує та оновлює інформацію про користувачів;
2. надає персоналізовані відповіді;
3. підтримує різні канали комунікації;
4. може розширювати функціонал без зміни базової архітектури.

Використання модульного підходу та стандартів REST API забезпечує масштабованість, гнучкість і сумісність системи з іншими програмними рішеннями, що є критично важливим для сучасних корпоративних чат-ботів.

2.5. Розробка моделі автомату станів

Кінцевий автомат (Finite State Machine, FSM) — це математична модель обчислень, що складається з кінцевої кількості станів, переходів між ними та дій. FSM є ефективним інструментом для моделювання поведінки діалогових систем, зокрема чат-ботів, оскільки дозволяє чітко структурувати логіку взаємодії з користувачем.

Формальне визначення:

$$M = (S, \Sigma, \delta, s_0, F)$$

де:

- **S** — кінцева множина станів системи
- **Σ** — вхідний алфавіт (множина можливих вхідних символів/команд)
- **δ: S × Σ → S** — функція переходів, що визначає наступний стан залежно від поточного стану та вхідного символу
- **$S_0 \in S$** — початковий стан системи
- **$F \subseteq S$** — множина кінцевих (допустимих) станів

2.5.2. Множина станів S

Таблиця 2.4 – Опис станів системи

Ідентифікатор	Назва стану	Функціональне призначення	Тип стану
S_0	Початковий стан	Ініціалізація діалогу, вітання користувача, пояснення можливостей системи	Початковий
S_1	Головне меню	Відображення основних функціональних опцій системи	Проміжний
S_2	Вибір категорії техніки	Класифікація техніки за типом (смартфони, ноутбуки, ПК, побутова техніка)	Проміжний
S_3	Опис проблеми	Збір текстового опису несправності від користувача	Проміжний
S_4	Завантаження фото	Опціональне отримання візуальних матеріалів пошкоджень (до 3 файлів)	Проміжний
S_5	Вибір дати і часу	Планування візиту клієнта для здачі техніки	Проміжний
S_6	Підтвердження заявки	Фінальна перевірка даних та створення запису в системі	Кінцевий
S_7	Перевірка статусу	Відстеження поточного стану ремонту за номером заявки	Кінцевий
S_8	База знань (FAQ)	Надання відповідей на типові питання користувачів	Кінцевий
S_9	Контактна інформація	Відображення адреси, телефону, режиму роботи	Кінцевий

2.5.3. Вхідний алфавіт Σ

Для формалізації роботи чат-бота в рамках скінченного автомата необхідно визначити вхідний алфавіт — множину всіх можливих сигналів (команд, повідомлень або дій користувача), що впливають на зміну станів системи. Вхідний алфавіт Σ описує типові дії користувача під час взаємодії з ботом і охоплює як кнопки меню, так і текстові повідомлення чи службові події («Назад», «Помилка», «Підтвердити» тощо). Чітке визначення елементів Σ забезпечує однозначність переходів між станами та дає змогу формально описати поведінку системи в різних сценаріях. Нижче наведено повний перелік вхідних символів, які обробляє система[18].

Вхідні команди користувача представлені множиною:

$$\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6, \sigma_7, \sigma_8, \sigma_9, \sigma_{10}, \sigma_{11}, \sigma_{12}, \sigma_{13}\}$$

де:

- σ_1 — команда /start (ініціалізація бота)
- σ_2 — "Створити заявку"
- σ_3 — вибір конкретної категорії техніки
- σ_4 — введення текстового опису проблеми
- σ_5 — завантаження фото
- σ_6 — "Без фото" / "Пропустити"
- σ_7 — вибір конкретної дати та часу
- σ_8 — "Підтвердити"
- σ_9 — "Статус ремонту"
- σ_{10} — "Часті питання"
- σ_{11} — "Контакти"
- σ_{12} — "Ціни"
- σ_{13} — "Назад" / "Головне меню"

Початковий стан:

$$s_0 = S_0$$

Множина кінцевих станів:

$$F = \{S_6, S_7, S_8, S_9, S_{10}\}$$

2.5.5. Матриця переходів станів

Для формалізації логіки роботи чат-бота використовується матриця переходів станів, яка відображає всі допустимі зміни між станами автомата залежно від отриманих вхідних символів. Такий підхід дозволяє чітко структурувати поведінку системи, уникнути неоднозначностей у трактуванні команд та забезпечити коректність обробки кожного запиту користувача. Матриця переходів також є основою для подальшого аналізу складності моделі, оцінювання навантаження та оптимізації діалогових сценаріїв. Узагальнену структуру переходів наведено у таблиці 2.5.

Таблиця 2.5 – Матриця переходів станів

Поточний стан (s_i)	**Вхідний символ (σ_j)**	**Наступний стан (s_k)**	**Дія системи**
S_0	σ_1 (/start)	S_1	Відобразити головне меню з основними опціями
S_1	σ_2 (Створити заявку)	S_2	Відобразити список категорій техніки
S_1	σ_9 (Статус ремонту)	S_7	Запитати номер заявки для перевірки
S_1	σ_{10} (FAQ)	S_8	Відобразити список частих питань
S_1	σ_{11} (Контакти)	S_9	Показати контактну інформацію сервісу
S_1	σ_{12} (Ціни)	S_{10}	Відобразити прайс-лист послуг
S_2	σ_3 (категорія)	S_3	Зберегти вибір, запитати опис проблеми
S_2	σ_{13} (Назад)	S_1	Повернутися до головного меню
S_3	σ_4 (текст)	S_4	Зберегти опис, запропонувати завантажити фото
S_3	σ_6 (Без фото)	**S_5**	Пропустити етап фото, перейти до вибору дати
S_3	σ_{13} (Назад)	**S_2**	Повернутися до вибору категорії
S_4	σ_5 (фото)	**S_5**	Зберегти зображення, запитати дату візиту
S_4	σ_6 (Пропустити)	**S_5**	Перейти до вибору дати без фото
S_4	σ_{13} (Назад)	**S_3**	Повернутися до введення опису
S_5	σ_7 (дата/час)	**S_6**	Зберегти час, показати підсумок заявки
S_5	σ_{13} (Назад)	**S_4**	Повернутися до завантаження фото
S_6	σ_8 (Підтвердити)	**S_1**	Створити запис у БД, видати номер заявки, повернутися до меню
S_6	σ_{13} (Змінити)	**S_5**	Повернутися до зміни дати/часу

Продовження таблиці 2.5

S_7	—	**S_1**	Відобразити статус заявки, повернутися до меню
S_8	σ_{13} (Головне меню)	**S_1**	Повернутися до головного меню
S_8	σ_2 (Створити заявку)	**S_2**	Перейти до створення заявки з FAQ
S_9	σ_{13} (Головне меню)	**S_1**	Повернутися до головного меню
S_{10}	σ_{13} (Головне меню)	**S_1**	Повернутися до головного меню
S_{10}	σ_2 (Створити заявку)	**S_2**	Перейти до створення заявки з прайсу

2.5.7. Аналіз коректності автомата

У межах побудованої моделі скінченного автомата проведено аналіз ключових сценаріїв обробки користувацьких запитів. Кожен сценарій характеризується послідовністю переходів між станами, що відображають типові взаємодії користувача з системою.

1) Основний сценарій(повний)

Маршрут переходів: $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5 \rightarrow S_6 \rightarrow S_1$

Послідовність етапів:

1. Ініціалізація системи командою /start.
2. Вибір користувачем опції «Створити заявку» у головному меню.
3. Вибір категорії техніки, що потребує ремонту.
4. Введення текстового опису проблеми.
5. Завантаження фотографії несправності.
6. Вибір дати та часу бажаного візиту майстра.
7. Підтвердження сформованої заявки.
8. Повернення до головного меню після успішного створення заявки.

Кількість переходів: 7.

2) Альтернативний сценарій(без фото)

Маршрут переходів: $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_5 \rightarrow S_6 \rightarrow S_1$

Цей сценарій передбачає пропуск етапу завантаження фотографії та використовується частиною користувачів, які не мають фото або бажають пришвидшити процес створення заявки.

Кількість переходів: 6.

3) Сценарій перевірки існуючої заявки

Маршрут переходів: $S_0 \rightarrow S_1 \rightarrow S_7 \rightarrow S_1$

Після вибору опції «Статус ремонту» система запитує номер заявки, виконує перевірку в базі даних та повертає результат користувачеві.

Кількість переходів: 3.
Для формальної оцінки поведінки побудованого автомата розглянуто такі характеристики:

1. Загальна кількість станів:
   |S|=11|S| = 11|S|=11
2. Кількість можливих переходів:
   |δ|=23|δ| = 23|δ|=23
   з них:
   - *прямі переходи (forward)* — 15;
   - *зворотні переходи (backward)* — 8.
3. Середня кількість вихідних переходів для одного стану:
   2311≈2.09\frac{23}{11} \approx 2.091123≈2.09
4. Максимальна глибина моделі (довжина найдовшого шляху від S_0):
   777
5. Кількість кінцевих станів:
   |F|=5|F| = 5|F|=5
   До кінцевих належать стани, у яких завершується логічний фрагмент сценарію, але не вся взаємодія (оскільки система циклічно повертає користувача до меню).
6. Наявність циклічності:
   Модель містить циклічні переходи, оскільки всі завершальні стани повертаються до S_1 (Головного меню). Це забезпечує безперервність взаємодії та можливість виконання декількох запитів у межах однієї сесії.

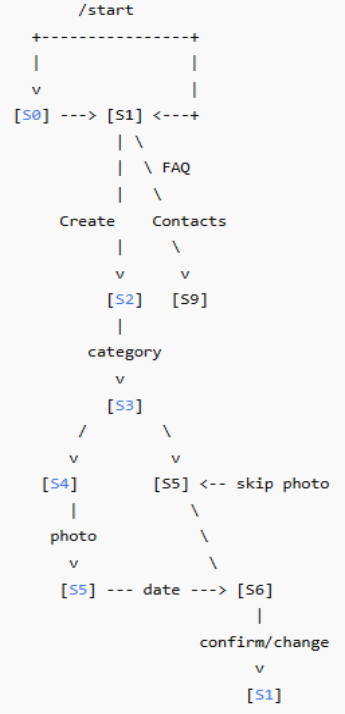


Рисунок 2.6 – Схема роботи автомату станів

Для оцінювання ефективності взаємодії користувача із системою проведено аналіз часових характеристик переходів між основними станами скінченного автомата. Такий аналіз дає змогу визначити середній час, необхідний для виконання кожної операції, а також мінімальні та максимальні значення затримок, що можуть виникати під час роботи. Отримані показники дозволяють оцінити зручність інтерфейсу, виявити потенційні «вузькі місця» та сформувати вимоги до оптимізації процесів. Зведені результати наведено у таблиці 2.6.

Таблиця 2.6 – Часові характеристики станів

Стан	Середній час (с)	Мінімум (с)	Максимум (с)	Основна дія користувача
S_0	5	2	10	Читання вітального повідомлення
S_1	3	1	8	Вибір опції з меню
S_2	5	2	15	Вибір категорії техніки
S_3	45	20	120	Набір текстового опису проблеми
S_4	30	10	90	Пошук та завантаження фото
S_5	15	5	45	Вибір дати та часу з календаря
S_6	10	3	30	Перегляд підсумку та підтвердження

Розрахунок загального часу проходження сценаріїв:

Повний сценарій (з фото):

$T\_повний = T\_S_0 + T\_S_1 + T\_S_2 + T\_S_3 + T\_S_4 + T\_S_5 + T\_S_6$

$T\_повний = 5 + 3 + 5 + 45 + 30 + 15 + 10 =$ **113 секунд ≈ 1 хв 53 с**

Скорочений сценарій (без фото):

$T\_скорочений = 5 + 3 + 5 + 45 + 15 + 10 =$ **83 секунди ≈ 1 хв 23 с**

Порівняння з ручною обробкою:

Час телефонного дзвінка до оператора: 180-300 с (3-5 хв) Економія часу: 67-117 секунд (37-56%)

Для підвищення точності моделювання поведінки користувачів та оптимізації логіки роботи чат-бота важливо враховувати імовірності переходів між станами автомата. Такі показники формуються на основі емпіричних даних або результатів тестових взаємодій і відображають реальні сценарії використання системи. Аналіз імовірностей дозволяє визначити найбільш типові маршрути, виявити рідкісні або проблемні переходи, а також оптимізувати інтерфейс та алгоритми обробки запитів. Узагальнені значення наведено у таблиці 2.7.

Таблиця 2.7 – Ймовірність переходів

Початковий стан	**Кінцевий стан**	**Ймовірність P**	**Емпіричне обґрунтування**
S_3	S_4	0.60	60% користувачів додають фото
S_3	S_5	0.40	40% пропускають завантаження
S_4	S_5 (з фото)	0.85	85% успішно завантажують фото
S_4	S_5 (пропустити)	0.15	15% відмовляються від фото
S_6	S_1 (підтвердити)	0.95	95% підтверджують заявку
S_6	S_5 (змінити)	0.05	5% змінюють дату/час
S_1	S_2	0.70	70% створюють нову заявку
S_1	S_7	0.15	15% перевіряють статус
S_1	S_8	0.08	8% переглядають FAQ
S_1	S_9	0.04	4% дивляться контакти
S_1	S_{10}	0.03	3% дивляться ціни

Розрахунок очікуваної ймовірності завершення заявки:

$P(\text{успіх}) = P(S_0 \to S_1) \times P(S_1 \to S_2) \times P(S_2 \to S_3) \times [P(S_3 \to S_4) \times P(S_4 \to S_5) + P(S_3 \to S_5)] \times P(S_5 \to S_6) \times P(S_6 \to S_1)$

$P(\text{успіх}) = 1 \times 0.7 \times 1 \times [0.6 \times 0.85 + 0.4] \times 1 \times 0.95$

$P(\text{успіх}) = 0.7 \times 0.91 \times 0.95 = \mathbf{0.605 \approx 60.5\%}$

Це означає, що приблизно 60.5% користувачів, які почали діалог, успішно створюють заявку[18].

Стратегії обробки некоректних входів:

1. **Невідома команда** — система залишається в поточному стані та відображає повідомлення з підказками
2. **Timeout (бездіяльність > 15 хвилин)** — автоматичне повернення до стану S_1 з збереженням контексту
3. **Некоректний формат даних** — запит повторного введення з поясненням очікуваного формату
4. **Системна помилка** — збереження стану сесії в БД, відображення вибачення, логування помилки

Таблиця 2.8 – Обробка вийняткових ситуацій

Ситуація	Поточний стан	Дія системи	Наступний стан
Невідома команда	Будь-який	Показати допоміжне повідомлення	Той самий
Timeout	S_2–S_6	Зберегти сесію, повідомити про очікування	S_1
Занадто велике фото	S_4	Запитати фото меншого розміру (<5MB)	S_4
Некоректний формат дати	S_5	Показати приклад формату	S_5
Відсутнє мережеве з’єднання	Будь-який	Чергувати повідомлення	Той самий
Переповнення бази даних	S_6	Повідомити про тимчасову недоступність	S_1

Висновок до розділу 2

У другому розділі було розглянуто основні технічні аспекти побудови системи автоматизації клієнтської підтримки на основі чат-бота. Здійснено порівняльний аналіз сучасних мов програмування та визначено, що Python є найбільш доцільною платформою для реалізації проєкту завдяки своїй гнучкості, наявності потужних бібліотек для обробки природної мови (NLP), підтримці API-інтеграцій і простоті розробки.

Було розроблено архітектуру системи, що ґрунтується на модульному підході та включає основні компоненти: модуль обробки запитів користувачів, модуль обробки природної мови, базу знань, підсистему інтеграції з базами даних і зовнішніми сервісами. Така структура забезпечує масштабованість, гнучкість і зручність оновлення системи без потреби суттєвих змін у базовій логіці.

У межах розділу детально описано алгоритм роботи чат-бота, який включає етапи отримання повідомлення, попередньої обробки тексту, визначення інтенції за допомогою NLP-моделі, пошуку відповіді у базі знань або звернення до великої мовної моделі (LLM), формування та надсилання відповіді користувачу. Алгоритм реалізує послідовну логіку діалогу, що дозволяє системі адаптуватися до контексту спілкування та підвищувати точність розпізнавання запитів.

Окрему увагу приділено інтеграції чат-бота з базами даних та зовнішніми сервісами. Реалізована структура забезпечує ефективну взаємодію між модулями системи, доступ до актуальної інформації про користувачів і динамічне оновлення бази знань. Крім того, використання REST API, Telegram API, CRM-систем і

аналітичних інструментів робить систему придатною для практичного впровадження в корпоративному середовищі.

Важливою перевагою розробленої архітектури є можливість самонавчання на основі накопичених історій діалогів і зворотного зв’язку від користувачів. Це створює умови для безперервного вдосконалення якості обслуговування та зниження навантаження на операторів підтримки.

Також було розроблено формальну модель кінцевого автомату. Модель включає 11 станів, 13 типів вхідних команд та 23 можливі переходи між станами.

Отже, у результаті проведеного аналізу та проєктування сформовано технічну основу системи автоматизації клієнтської підтримки, яка поєднує інтелектуальні методи обробки тексту, модульну архітектуру та інтеграцію з зовнішніми сервісами. Це забезпечує високу ефективність, надійність і адаптивність системи, що відповідає вимогам сучасних інформаційних технологій.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ АВТОМАТИЗОВАНОЇ КЛІЄНТСЬКОЇ ПІДТРИМКИ

Практична частина роботи присвячена безпосередній розробці та впровадженню системи чат-бота для автоматизації клієнтської підтримки сервісного центру. У цьому розділі наведено опис вибору технологій, структури програмних модулів, механізмів взаємодії з базою даних, а також реалізації ключових компонентів — обробника повідомлень, модуля NLP, сценарної логіки та інтерфейсу для кінцевих користувачів.

Метою практичної реалізації є створення працездатного, масштабованого та зручного у використанні програмного рішення, що забезпечує:

автоматизовану обробку заявок користувачів;

формування структурованих записів у базі даних;

інтеграцію з Telegram-платформою через Bot API;

підтримку сценарної логіки та переходів між станами;

можливість подальшого розширення функціональності за рахунок модульної архітектури.

У межах цього розділу детально описано розроблені програмні модулі, їх взаємодію, методи тестування та особливості програмної реалізації алгоритмів, сформованих у теоретичній частині. Також розглядаються аспекти оптимізації роботи чат-бота, обробки помилок і забезпечення стабільності системи в реальних умовах експлуатації.

3.1. Підготовка середовища розробки та інструментів

Оскільки у другому розділі було обґрунтовано вибір технологічного стека — мови програмування Python та середовища розробки Visual Studio Code, на практичному етапі виконується налаштування інструментів, необхідних для реалізації чат-бота.

Для розробки програмної частини системи було використано VS Code, що забезпечує зручність роботи з Python-кодом, підтримку численних розширень, інтелектуальну підсвітку синтаксису, автозаповнення та інтегровану систему керування середовищами. Завдяки цьому VS Code виступає оптимальним інструментом для створення модульної архітектури чат-бота та забезпечує швидку навігацію між компонентами проєкту.

Пакетний менеджер pip використовувався для встановлення необхідних бібліотек, зокрема:

python-telegram-bot — основна бібліотека для взаємодії з Telegram Bot API;

sqlite3 / SQLAlchemy — для побудови бази даних та ORM-рівня;

nltk, spaCy або інші NLP-інструменти — для попередньої обробки тексту;

Pillow — для роботи з графічними файлами, якщо потрібно обробляти фотографії;

dotenv — для безпечного зберігання токена бота;

logging — для створення журнальних записів та діагностики системи.

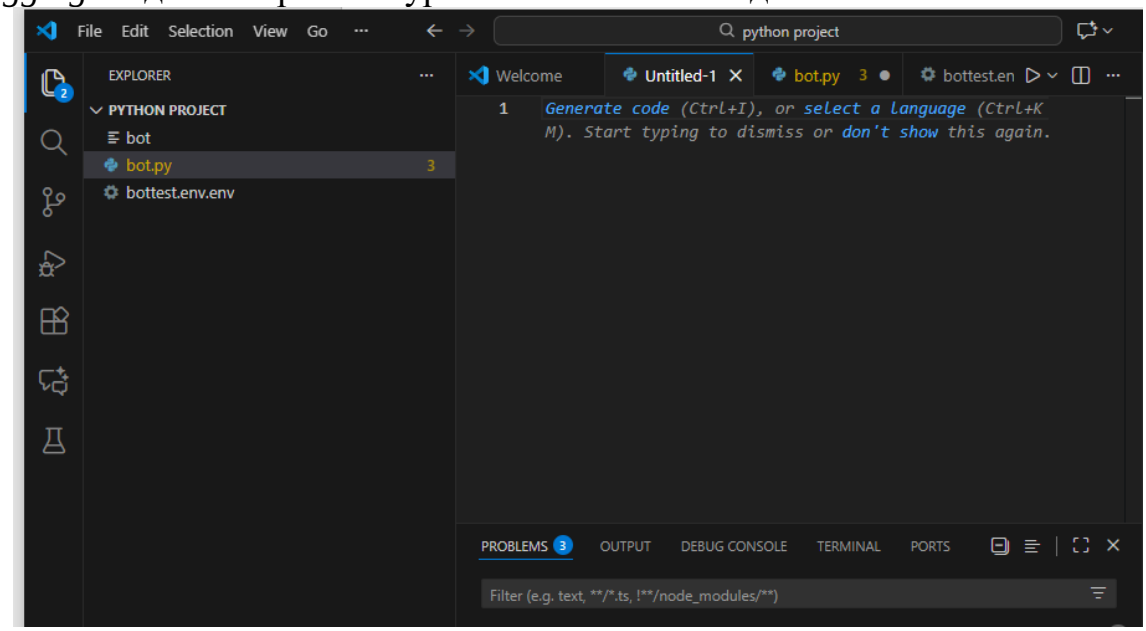


Рисунок 3.1 – Інтерфейс VS code

Для ізоляції залежностей було створено віртуальне середовище:

python -m venv venv

venv\Scripts\activate (Windows)

Архітектура проєкту організована у вигляді кількох окремих модулів:

bot/ — логіка Telegram-бота;

handlers/ — обробники повідомлень і команд;

database/ — взаємодія з базою даних, таблиці заявок;

nlu/ — модулі NLP, попередня обробка тексту;

states/ — реалізація моделі станів;

config/ — файли конфігурацій і ключів;

utils/ — допоміжні функції.

Такий підхід дозволив структурувати застосунок таким чином, щоб його було легко підтримувати, розширювати та інтегрувати з додатковими сервісами у майбутньому.

3.2. Структура програмного проєкту та опис основних модулів

Програмна реалізація чат-бота побудована за модульним принципом, що забезпечує розділення відповідальностей, можливість подальшого масштабування та зручність підтримки. Усі компоненти системи згруповані в

окремі каталоги, кожний з яких відповідає за конкретний функціональний аспект роботи бота.

```
project/
|
├── bot/
|   ├── main.py
|   ├── dispatcher.py
|   ├── keyboards.py
|   └── callbacks.py
|
├── states/
|   ├── fsm_states.py
|   └── transitions.py
|
├── handlers/
|   ├── start_handler.py
|   ├── application_handler.py
|   ├── status_handler.py
|   ├── faq_handler.py
|   └── error_handler.py
|
├── database/
|   ├── database.py
|   ├── models.py
|   └── queries.py
|
├── nlu/
|   ├── text_preprocessing.py
|   ├── intent_classifier.py
|   └── synonyms.py
|
├── utils/
|   ├── validators.py
|   ├── file_processing.py
|   └── logging_config.py
|
├── config/
|   ├── config.py
|   └── .env
|
└── requirements.txt
```

Рисунок 3.2 – Структура проєкту

Модуль bot

Модуль bot відповідає за взаємодію з Telegram Bot API та забезпечує: ініціалізацію бота і запуск механізму обробки подій; маршрутизацію вхідних повідомлень; формування та відправлення відповідей користувачам; генерацію клавіатур та елементів інтерфейсу.

Основні файли модуля:

main.py — точка входу до системи, запуск основного циклу обробки повідомлень;

dispatcher.py — реєстрація обробників команд та повідомлень;

keyboards.py — формування кнопок меню та інтерактивних елементів;

callbacks.py — обробка callback-подій, що надходять від інлайн-кнопок.

Модуль states

Модуль states реалізує модель скінченного автомата, описану в теоретичному розділі. Основне призначення модуля — забезпечення коректної логіки навігації між станами діалогу.

До складу модуля входять:

fsm_states.py — визначення множини станів S (S_0–S_{10});

transitions.py — реалізація функції переходів δ та сценарних правил діалогової логіки.

Модуль забезпечує детермінованість переходів і запобігає виникненню некоректних станів під час взаємодії з користувачем.

Модуль handlers

Модуль handlers містить спеціалізовані обробники, що відповідають за реакцію системи на конкретні дії користувача. Кожний файл модуля реалізує окремий функціональний блок обслуговування запитів[15].

Основні елементи модуля:

start_handler.py — обробка команди /start, формування привітального повідомлення;

application_handler.py — реалізація логіки створення заявки та взаємодії з підмодулем переходів станів;

status_handler.py — виконання запиту перевірки статусу ремонту;

faq_handler.py — обробка звернень до блоку частих питань (FAQ);

error_handler.py — обробка помилкових ситуацій, логування помилок і відображення службових повідомлень.

Модуль реалізує основну бізнес-логіку системи.

Модуль database

Модуль nlu (Natural Language Understanding) відповідає за елементи попередньої обробки текстових повідомлень та розпізнавання інтенцій користувачів.

До основних компонентів належать:

text_preprocessing.py — очистка тексту, нормалізація, токенізація;

intent_classifier.py — класифікація інтенцій користувача на основі ключових слів, шаблонів або моделей ML;

synonyms.py — словник синонімів та розширених словоформ, що підвищують точність класифікації.

Модуль забезпечує інтелектуальні можливості чат-бота та покращує взаємодію з користувачем у текстовій формі.

Модуль utils

Модуль utils містить допоміжні функції, необхідні для стабільної роботи системи:

validators.py — перевірка коректності введених даних (дати, розміру фото, формату повідомлення);

file_processing.py — обробка файлів, зокрема перевірка розширення та масштабування зображень;

logging_config.py — налаштування логування системи, формування журналу подій.

Модуль сприяє підвищенню надійності та відмовостійкості програмного забезпечення.

Модуль config

Модуль config містить конфігураційні файли та параметри, необхідні для роботи всієї системи:

config.py — загальні налаштування бота, шляхи до ресурсів, налаштування БД;

.env — файл із секретними даними (токен Telegram-бота), що не включається до публічного репозиторію.

Централізація конфігурації полегшує супровід системи та забезпечує безпечне зберігання чутливої інформації.

3.3. Реалізація модуля обробки повідомлень

Модуль обробки повідомлень є центральним елементом практичної реалізації чат-бота, оскільки забезпечує отримання, аналіз і маршрутизацію всіх вхідних дій користувачів. У межах цього модуля реалізовано функціональність визначення типу повідомлення, виклику відповідного обробника та формування відповіді відповідно до поточного стану діалогу.

Основою роботи модуля є використання бібліотеки **python-telegram-bot**, яка забезпечує зручний інтерфейс для взаємодії з Telegram Bot API. Система застосовує підхід *event-driven*, тобто кожне повідомлення користувача автоматично викликає певний обробник, зареєстрований у диспетчері.

3.3.1. Принцип роботи диспетчера

Реалізовано диспетчер подій, що виконує такі функції: приймає повідомлення від Telegram API; класифікує повідомлення за типом (команда,

текст, зображення, callback); визначає поточний стан користувача у FSM; викликає відповідний обробник із модуля *handlers*; виконує перехід до наступного стану згідно з моделлю δ; формує відповідь і надсилає її користувачеві.

```
from telegram.ext import ApplicationBuilder, CommandHandler, MessageHandler, filters
from handlers.start_handler import start
from handlers.application_handler import handle_application
from handlers.status_handler import handle_status

app = ApplicationBuilder().token(BOT_TOKEN).build()

app.add_handler(CommandHandler("start", start))
app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, handle_application))
app.add_handler(MessageHandler(filters.PHOTO, handle_application))
app.add_handler(MessageHandler(filters.COMMAND, error_handler))
```

Рисунок 3.3 – Структура ініціалізації диспетчера

3.3.2. Обробка команди /start

Команда */start* ініціалізує діалог, встановлює початковий стан S_0 та відображає головне меню. Логіка обробника: реєстрація користувача в базі (якщо він новий); скидання попереднього стану; надсилання привітального повідомлення; перехід до стану S_1.

```
def start(update, context):
    user_id = update.effective_user.id
    set_state(user_id, "S1")
    update.message.reply_text(
        "Вітаємо! Оберіть одну з опцій нижче:",
        reply_markup=main_menu_keyboard()
    )
```

Рисунок 3.4 – Фрагмент коду для обробки команди

3.3.3. Обробка текстових повідомлень

Текстові повідомлення обробляються залежно від поточного стану користувача: у стані S_3 — запис текстового опису проблеми; у стані S_7 — введення номера заявки; у станах FAQ або меню — фрази інтерпретуються через модуль NLP.

Спрощений фрагмент логіки обробника:

```python
def handle_text(update, context):
    user_id = update.effective_user.id
    state = get_state(user_id)

    if state == "S3":
        save_problem_description(user_id, update.message.text)
        set_state(user_id, "S4")
        update.message.reply_text("Дякуємо! Тепер завантажте фото або натисніть 'Пропустити'.")
```

Рисунок 3.5 – Спрощений фрагмент логіки обробника

3.3.4. Обробка фотографій

У стані **S_4** користувач може надіслати фотографію несправності. Виконується: перевірка розміру та формату файлу; збереження фото у файловій системі або БД; перехід до стану **S_5**.

```python
def handle_photo(update, context):
    user_id = update.effective_user.id
    state = get_state(user_id)

    if state == "S4":
        photo = update.message.photo[-1]
        file_id = photo.file_id
        save_photo(user_id, file_id)
        set_state(user_id, "S5")
        update.message.reply_text("Фото збережено. Оберіть дату та час візиту.")
```

Рисунок 3.6 – Обробка фотографій

3.3.5. Обробка callback-натискань

Callback-кнопки використовуються для: навігації «Назад»; вибору категорії техніки; переходу між меню; підтвердження заявки.

```python
def handle_callback(update, context):
    query = update.callback_query
    action = query.data

    if action == "confirm":
        save_application(query.from_user.id)
        set_state(query.from_user.id, "S1")
        query.edit_message_text("Заявку створено успішно!")
```

Рисунок 3.7 – Callback-кнопки

3.4. Реалізація моделі станів у програмному коді

У цьому підрозділі розглядається практична реалізація моделі скінченного автомата (FSM), що визначає логіку роботи чат-бота. Модель станів, розроблена на теоретичному етапі, формалізує взаємодію користувача з системою, забезпечує детерміновані переходи між станами та визначає реакцію бота на відповідні команди.

У програмній реалізації FSM використано словникову структуру (dict), що дає змогу зберігати правила переходів у компактній та зручній формі. Такий підхід забезпечує:

простоту додавання нових станів та команд;
прозору логіку переходів;
можливість автоматизованого тестування;
відсутність дублювання коду;
відповідність моделі формальному опису δ:

```
class States:
    S0 = "S0"
    S1 = "S1"
    S2 = "S2"
    S3 = "S3"
    S4 = "S4"
    S5 = "S5"
    S6 = "S6"
    S7 = "S7"
    S8 = "S8"
    S9 = "S9"
    S10 = "S10"
```

Рисунок 3.7 – Створення станів

```
class Commands:
    START = "σ1"
    CREATE = "σ2"
    CATEGORY = "σ3"
    TEXT = "σ4"
    PHOTO = "σ5"
    SKIP = "σ6"
    DATETIME = "σ7"
    CONFIRM = "σ8"
    STATUS = "σ9"
    FAQ = "σ10"
    CONTACTS = "σ11"
    PRICES = "σ12"
    BACK = "σ13"
```

Рисунок 3.8 – Команди відповідно до Σ

Далі на потрібно реалізуватии нашу матрицю станів (таб. 2.5). Це можна зробити наступним чиином:

```
TRANSITIONS = {
    States.S0: {
        Commands.START: States.S1
    },

    States.S1: {
        Commands.CREATE: States.S2,
        Commands.STATUS: States.S7,
        Commands.FAQ: States.S8,
        Commands.CONTACTS: States.S9,
        Commands.PRICES: States.S10
    },

    States.S2: {
        Commands.CATEGORY: States.S3,
        Commands.BACK: States.S1
    },

    States.S3: {
        Commands.TEXT: States.S4,
        Commands.SKIP: States.S5,
        Commands.BACK: States.S2
    },

    States.S4: {
        Commands.PHOTO: States.S5,
        Commands.SKIP: States.S5,
        Commands.BACK: States.S3
    },

    States.S5: {
        Commands.DATETIME: States.S6,
        Commands.BACK: States.S4
    },

    States.S6: {
        Commands.CONFIRM: States.S1,
        Commands.BACK: States.S5
    },

    States.S7: {
        # Після показу статусу завжди повертається до головного меню
        None: States.S1
    },

    States.S8: {
        Commands.BACK: States.S1,
        Commands.CREATE: States.S2
    },

    States.S9: {
        Commands.BACK: States.S1
    },

    States.S10: {
        Commands.BACK: States.S1,
        Commands.CREATE: States.S2
    }
}
```

Рисунок 3.9 – Реалізація матриці преходів

```python
def next_state(current_state, command):
    transitions = TRANSITIONS.get(current_state, {})

    # якщо команда відсутня – залишаємось у поточному стані
    return transitions.get(command, current_state)
```

Рисунок 3.10 – Функція переходу δ

Створення структури бази даних здійснюється у функції init_db():

```python
def init_db():
    conn = sqlite3.connect(DB_NAME)
    cur = conn.cursor()
    cur.execute(
        """
        CREATE TABLE IF NOT EXISTS applications (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER,
            username TEXT,
            category TEXT,
            description TEXT,
            photo_file_id TEXT,
            visit_datetime TEXT,
            status TEXT DEFAULT 'Створена',
            created_at TEXT
        )
        """
    )
    conn.commit()
    conn.close()
```

Рисунок 3.11 – Ініціалізація бази даних

sqlite3.connect(DB_NAME) – відкриває (або створює) файл бази даних applications.db.

CREATE TABLE IF NOT EXISTS applications – створює таблицю заявок, якщо вона ще не існує.

Поля таблиці:

id – первинний ключ (унікальний номер заявки).

user_id – ідентифікатор користувача Telegram.

username – ім’я користувача (нікнейм).

category – обрана категорія техніки.

description – текстовий опис проблеми.

photo_file_id – ідентифікатор фото в Telegram (для можливості подальшого завантаження з серверів Telegram).

visit_datetime – запланована дата та час візиту.

status – статус заявки (за замовчуванням – 'Створена').

created_at – дата й час створення заявки.

Після виконання SQL-команди зміни фіксуються (commit) і з'єднання з базою закривається.

Для зручності користувача використовуються **Reply-клавіатури**, які дозволяють обирати готові кнопки, а не вводити текст вручну.

```
def main_menu_keyboard():
    kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
    kb.row(types.KeyboardButton("Створити заявку"),
           types.KeyboardButton("Статус ремонту"))
    kb.row(types.KeyboardButton("FAQ"),
           types.KeyboardButton("Контакти"),
           types.KeyboardButton("Ціни"))
    return kb
```

Рисунок 3.12 – Головне меню

Клавіатура головного меню має дві строки:

перший рядок: «Створити заявку», «Статус ремонту»;

другий рядок: «FAQ», «Контакти», «Ціни».

Параметр resize_keyboard=True робить клавіатуру компактною й адаптованою під мобільний екран.

```
def back_keyboard():
    kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
    kb.row(types.KeyboardButton("Назад"),
           types.KeyboardButton("Головне меню"))
    return kb
```

Рисунок 3.13 – Кнопки «Назад» та «Головне меню»

Використовується на проміжних кроках, коли користувач може повернутися на попередній етап або у головне меню.

```
def skip_or_back_keyboard():
    kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
    kb.row(types.KeyboardButton("Пропустити"))
    kb.row(types.KeyboardButton("Назад"),
           types.KeyboardButton("Головне меню"))
    return kb
```

Рисунок 3.14 – Клавіатура з можливістю пропустити фото

Застосовується на етапі S4 (завантаження фото): користувач може надіслати фото або натиснути «Пропустити».

```
def confirm_keyboard():
    kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
    kb.row(types.KeyboardButton("Підтвердити"))
    kb.row(types.KeyboardButton("Змінити дату/час"))
    kb.row(types.KeyboardButton("Головне меню"))
    return kb
```

Рисунок 3.15 – Клавіатура підтвердження заявки

На етапі підтвердження S6 користувач або погоджується із заявкою, або повертається до зміни дати/часу, або виходить у головне меню.

```
def categories_keyboard():
    kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
    kb.row(types.KeyboardButton("Смартфон"),
           types.KeyboardButton("Ноутбук"))
    kb.row(types.KeyboardButton("ПК"),
           types.KeyboardButton("Побутова техніка"))
    kb.row(types.KeyboardButton("Назад"),
           types.KeyboardButton("Головне меню"))
    return kb
```

Рисунок 3.16 – Клавіатура вибору категорії

Дозволяє користувачеві вибрати тип техніки, для якої створюється заявка.

```
@bot.message_handler(commands=["start"])
def cmd_start(message: types.Message):
    user_id = message.from_user.id
    set_state(user_id, States.S1)
    bot.send_message(
        message.chat.id,
        "Вітаємо у сервісному чат-боті!\n"
        "Через цього бота ви можете створити заявку на ремонт, "
        "перевірити статус, переглянути FAQ, контакти та ціни.",
        reply_markup=main_menu_keyboard(),
    )
```

Рисунок 3.17 – Обробник команди старт

Декоратор @bot.message_handler(commands=["start"]) реєструє функцію як обробник команди /start.

При першому запуску бота користувачеві встановлюється стан S1 (головне меню).

Надсилається привітальне повідомлення з поясненням можливостей та клавіатурою головного меню.

```
if text.lower() == "головне меню":
    set_state(user_id, States.S1)
    bot.send_message(
        message.chat.id,
        "Повернення до головного меню.",
        reply_markup=main_menu_keyboard(),
    )
    return
```

Рисунок 3.18 – Команда головне меню

Незалежно від поточного стану, якщо користувач вводить «Головне меню», його автоматично повертають до стану S1, а на екран виводиться головна клавіатура. Це забезпечує зручний «аварійний» вихід із будь-якого етапу.

```
if state == States.S1:
    if text == "Створити заявку":
        ...
    elif text == "Статус ремонту":
        ...
    elif text == "FAQ":
        ...
    elif text == "Контакти":
        ...
    elif text == "Ціни":
        ...
    else:
        ...
    return
```

Рисунок 3.19 – Логіка S1

У цьому стані розглядаються п’ять основних дій:

1. «Створити заявку»

Перехід до стану S2.

Вивід клавіатури вибору категорії:

```
set_state(user_id, States.S2)
bot.send_message(
    message.chat.id,
    "Оберіть категорію техніки:",
    reply_markup=categories_keyboard(),
)
```

Рисунок 3.20 – Створити заявку

2. «Статус ремонту»

Перехід до стану S7.

Запит номера заявки у текстовому форматі:

```
set_state(user_id, States.S7)
bot.send_message(
    message.chat.id,
    "Введіть, будь ласка, номер вашої заявки:",
    reply_markup=back_keyboard(),
)
```

Рисунок 3.21 – Статус ремонту

3. «FAQ»

Перехід у стан S8.

Виведення переліку типових питань та кнопки повернення в головне меню.

4. «Контакти» (стан S9) – виводиться адреса, телефон та графік роботи сервісного центру.
5. «Ціни» (стан S10) – виводяться орієнтовні вартісні діапазони послуг.

У випадку невідомої команди користувачу надсилається повідомлення про помилку та повторно показується головне меню.

У цьому стані користувач обирає тип техніки:

При натисканні «Назад» – повернення у головне меню:

```
if text == "Назад":
    set_state(user_id, States.S1)
    ...
    return
```

Рисунок 3.22 – Логіка S2

Якщо текст входить у список категорій ["Смартфон", "Ноутбук", "ПК", "Побутова техніка"]:

Значення зберігається в user_data[user_id]["category"].

Стан змінюється на S3 (опис проблеми).

Користувачу пропонується ввести текстовий опис несправності.

Якщо введено інший текст – користувачеві надсилається нагадування обрати категорію саме з клавіатури.

У стані S3 користувач вводить текстовий опис несправності:

При натисканні «Назад» – повернення до вибору категорії S2.

У будь-якому іншому випадку введений текст зберігається як description:

```
data = user_data.setdefault(user_id, {})
data["description"] = text
```

Рисунок 3.23 – Логіка S3

Після цього бот переходить до стану S4 (етап завантаження фото) і повідомляє, що можна надіслати фото або натиснути «Пропустити».

У стані S4 можливі такі варіанти:

Користувач натискає «Назад» – повернення до опису проблеми (S3).

Користувач натискає «Пропустити» – в user_data поле photo_file_id встановлюється в None, і відбувається перехід до стану S5 (вибір дати/часу).

Якщо користувач надсилає текст замість фото та не обирає «Пропустити», бот нагадує про можливість пропуску фото.

Зверніть увагу: обробка реальних фото відбувається у **відокремленому обробнику** handle_photo.

У стані S5 користувач вводить бажану дату й час візиту у форматі ДД.ММ.РРРР ГГ:ХХ.

При натисканні «Назад» бот повертається до стану S4.

Інакше бот намагається розпарсити рядок:

```
dt = datetime.strptime(text, "%d.%m.%Y %H:%M")
data["visit_datetime"] = dt.strftime("%d.%m.%Y %H:%M")
```

Рисунок 3.23 – Логіка S5

Якщо формат некоректний, користувачу відправляється повідомлення з прикладом правильного введення.

У разі успішного розпізнавання здійснюється перехід до стану S6 – підтвердження заявки. Користувачу показуються всі введені дані (категорія, опис, дата й час) і пропонується їх підтвердити.

У стані S6 користувач може:

1. **Підтвердити заявку («Підтвердити»)**

   У цьому випадку:

   з user_data зчитуються всі необхідні поля;

   відкривається з’єднання з базою даних SQLite;

   виконується SQL-операція вставки нового запису:

```
cur.execute(
    """
    INSERT INTO applications
    (user_id, username, category, description, photo_file_id, visit_datetime, status, creat
    VALUES (?, ?, ?, ?, ?, ?, 'Створена', ?)
    """,
    (
        user_id,
        user.username,
        category,
        desc,
        photo_id,
        dt_str,
        datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
    ),
)
```

Рисунок 3.24 – Логіка S6

cur.lastrowid зберігає згенерований номер заявки app_id.

Після commit() з'єднання із базою закривається.

Стан користувача повертається до S1, йому надсилається підтвердження із номером заявки, який надалі можна використати для перевірки статусу.

1. **Змінити дату/час («Змінити дату/час»)**

   Стан змінюється назад на S5.

   Користувачеві пропонують знову ввести дату та час візиту.

2. **Будь-який інший текст**

   Користувачеві повторно пропонують обрати одну з доступних опцій («Підтвердити» або «Змінити дату/час»).

   У стані S7 очікується введення **номера заявки**:

   При натисканні «Назад» – повернення до S1.

   Якщо введене значення не є числом (ValueError при перетворенні int(text)), відображається повідомлення про помилку.

   У разі коректного номера app_id виконується SQL-запит:

```
cur.execute(
    "SELECT status, category, visit_datetime FROM applications WHERE id = ?",
    (app_id,),
)
```

Рисунок 3.25 – Логіка S7

Якщо заявка знайдена, користувачу повертається інформація про категорію, дату візиту та поточний статус.

Якщо заявки з таким номером немає – бот повідомляє, що запис не знайдено.

Після відповіді користувачеві стан повертається до S1.

Для станів FAQ (S8), Контакти (S9) і Ціни (S10) логіка однакова: будь-яке введене текстове повідомлення призводить до виводу інструкції повернутися у головне меню:

```
if state in [States.S8, States.S9, States.S10]:
    ...
    bot.send_message(
        message.chat.id,
        "Оберіть «Головне меню» для продовження роботи.",
        reply_markup=kb,
    )
    return
```

Рисунок 3.26 – Логіка S8,S9,S10

Окремий обробник відповідає за обробку повідомлень із типом photo:

```
@bot.message_handler(content_types=["photo"])
def handle_photo(message: types.Message):
    user_id = message.from_user.id
    state = get_state(user_id)

    if state != States.S4:
        ...
        return

    photo = message.photo[-1]
    file_id = photo.file_id
    data = user_data.setdefault(user_id, {})
    data["photo_file_id"] = file_id

    set_state(user_id, States.S5)
    bot.send_message(
        message.chat.id,
        "Фото збережено. Введіть дату та час візиту (формат: ДД.ММ.РРРР ГГ:ХХ):",
        reply_markup=back_keyboard(),
    )
```

Рисунок 3.27 – Обробка фото

Якщо поточний стан користувача не S4, бот повідомляє, що фото можна надсилати лише на етапі оформлення заявки, і повертає клавіатуру головного меню. Це запобігає некоректному додаванню зображень у невідповідний момент діалогу.

Якщо стан – S4, з масиву message.photo береться останній (найбільший за розміром) об’єкт фото. З нього зчитується file_id, який зберігається у user_data як photo_file_id. Далі бот переходить до стану S5 та просить ввести дату й час візиту.

Таким чином, фото прив’язується до поточної заявки, але фізично не зберігається в базі даних; зберігається лише file_id Telegram, що у разі потреби дозволяє завантажити файли з сервера Telegram.

Наприкінці файлу визначена стандартна точка входу:

```
if __name__ == "__main__":
    init_db()
    print("Бот запущено. Відкрий Telegram і пройди сценарій створення заявки.")
    bot.infinity_polling()
```

Рисунок 3.28 – Точка входу

Умова if __name__ == "__main__": гарантує, що код у блоці буде виконано лише при безпосередньому запуску цього файлу як основної програми, а не при імпорті як модуля.

Спочатку викликається init_db(), що забезпечує готовність бази даних (створення таблиці applications при необхідності).

Далі виводиться службове повідомлення у стандартний потік виводу (консоль).

Метод bot.infinity_polling() запускає безперервний цикл опитування серверів Telegram: бот отримує нові оновлення (повідомлення), викликає відповідні обробники та працює доти, доки процес не буде зупинено.

У результаті розробки програмного коду сервісного чат-бота на платформі Telegram було створено приклад прикладного програмного забезпечення, яке реалізує повноцінний цикл обробки клієнтської заявки на ремонт техніки – від початкового звернення користувача до збереження структурованих даних у базі та можливості подальшого перегляду статусу.

Архітектура чат-бота побудована на поєднанні кількох ключових технічних рішень:

FSM-підхід (кінцевий автомат станів), логіка діалогу реалізована у вигляді набору дискретних станів (S1–S10), кожен з яких відповідає певному етапу сценарію: головне меню, вибір категорії, опис проблеми, завантаження фото, вибір дати та часу, підтвердження заявки, перевірка статусу, інформаційні режими FAQ/контакти/прайс. Такий підхід: абезпечує передбачуваність та керованість діалогу; дає змогу легко відслідковувати поточний етап взаємодії для кожного окремого користувача; спрощує подальше розширення сценарію (додавання нових кроків, опцій, гілок діалогу) [19].

Зберігання проміжних даних користувача в оперативній пам’яті Для кожного користувача ведеться власний словник даних (user_data) та поточний стан (user_states), що дозволяє: поступово формувати заявку, поетапно додаючи

до неї категорію, опис, фото та дату візиту; реалізувати можливість повернення «Назад» на попередні кроки без втрати вже введеної інформації; одночасно обслуговувати множину користувачів, які можуть перебувати на різних етапах оформлення заявки.

Запропонована структура таблиці applications є достатньо гнучкою та містить усі ключові атрибути заявки: ідентифікатор, дані користувача, категорію, опис, ідентифікатор фото, час запланованого візиту, статус та дату створення. Це забезпечує: надійне довготривале зберігання інформації; можливість подальшої обробки заявок персоналом сервісного центру; можливість розширення функціоналу (наприклад, додавання полів про вартість, відповідального майстра, етапи ремонту).

Інтеграція з Telegram через бібліотеку telebot Використання бібліотеки pyTelegramBotAPI (telebot) дозволило: реалізувати обробку команд (/start) і різних типів повідомлень (текст, фото); побудувати зручний інтерфейс на основі Reply-клавіатур (головне меню, вибір категорії, підтвердження тощо); працювати з фото не як з «важкими» файлами, а як з ідентифікаторами file_id, що у разі потреби можуть бути використані для повторного доступу до зображень на серверах Telegram. Валідація та обробка помилкових дій користувача У коді передбачено базові механізми валідації: перевірка коректності формату дати та часу (ДД.ММ.РРРР ГГ:ХХ); перевірка числового формату номера заявки; обробка ситуацій, коли користувач вводить невідомі команди або знаходиться в некоректному стані. Це підвищує надійність системи, зменшує ймовірність помилок користувача та покращує загальний користувацький досвід.

Окрім створення заявок, бот надає додаткові довідкові можливості: FAQ, контакти сервісного центру та орієнтовний прайс. Завдяки цьому взаємодія користувача з сервісом відбувається в одному вікні Telegram, без потреби звертатися до сайту чи телефонувати в call-центр для отримання базової інформації.

3.5. Тестування чат боту

Тестування є завершальним етапом розроблення програмного забезпечення і має на меті перевірку відповідності реалізованої системи заданим вимогам. Для чат-бота клієнтської підтримки це, насамперед, коректність оброблення діалогових сценаріїв, надійність роботи зі сховищем даних та зручність взаємодії для кінцевого користувача.

Розроблений чат-бот було протестовано в умовах, максимально наближених до реальної експлуатації. Тестування проводилося на персональному комп’ютері під керуванням операційної системи Windows, із встановленим інтерпретатором Python, бібліотекою pyTelegramBotAPI та локальною базою даних SQLite. Для взаємодії використовувався офіційний клієнт Telegram (мобільний та десктопний).

3.5.1 Мета та завдання тестування

Основною метою тестування є перевірка того, що чат-бот:

правильно реалізує усі основні сценарії взаємодії, описані у моделі скінченного автомата;

коректно зберігає дані у базі applications.db та надає доступ до них під час перевірки статусу;

стійко поводиться за некоректного введення користувачем (помилковий формат дати, невірний номер заявки, непередбачені команди);

забезпечує прийнятний час реакції на повідомлення користувача.

Для цього були сформовані наступні групи тестових завдань:

Функціональне тестування основних сценаріїв:

створення заявки з фото;

створення заявки без фото (із пропуском етапу завантаження зображення);

перегляд статусу наявної заявки;

робота розділів «FAQ», «Контакти», «Ціни».

Перевірка помилкових та граничних ситуацій:

введення дати у неправильному форматі;

введення нечислового значення замість номера заявки;

надсилання фото в «неправильному» стані (коли бот не очікує зображення);

використання невідомих команд та довільного тексту в головному меню.

Перевірка коректності роботи БД:

створення нового запису заявки;

зчитування запису за ідентифікатором;

поведінка у випадку запиту неіснуючого номера заявки.

3.5.2 Методика функціонального тестування

Функціональне тестування виконувалося у вигляді послідовного проходження користувачем усіх сценаріїв, передбачених моделлю станів.

Сценарій створення заявки з фото

Команда /start або натискання кнопки «Створити заявку» в головному меню.

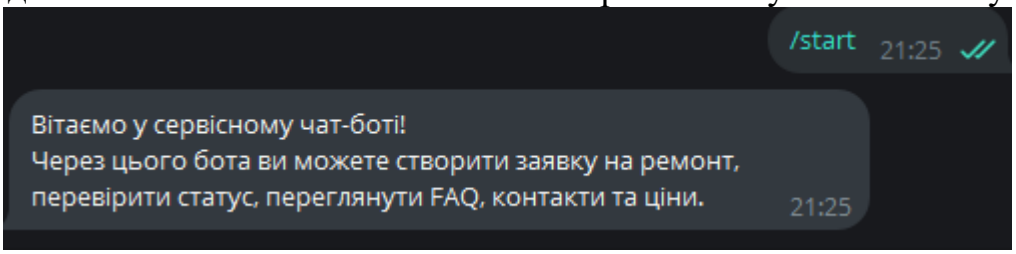


Рисунок 3.29 – Крок 1

Вибір однієї з категорій («Смартфон», «Ноутбук», «ПК», «Побутова техніка»).

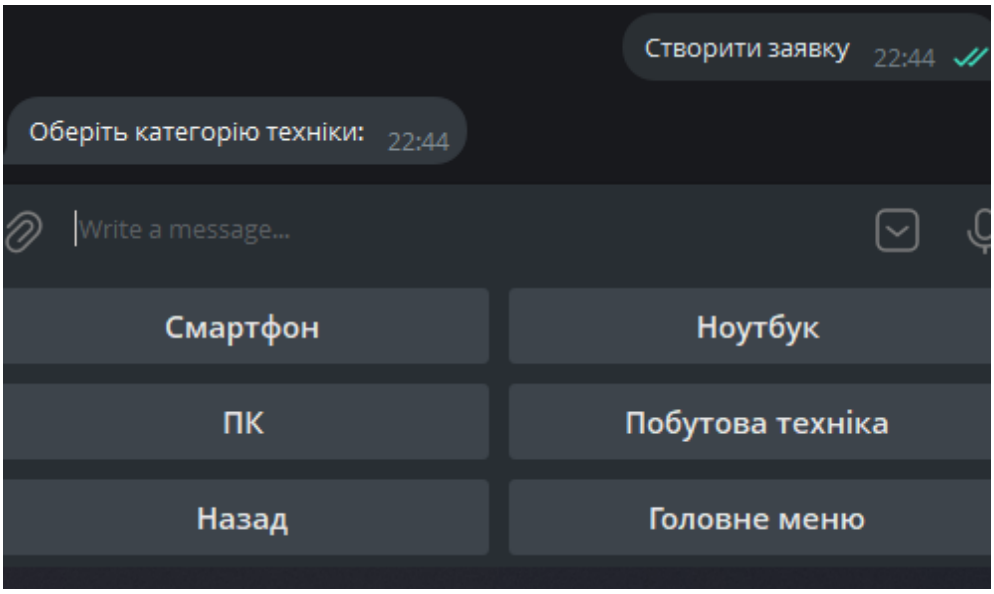


Рисунок 3.30 – Крок 2

Введення текстового опису проблеми.

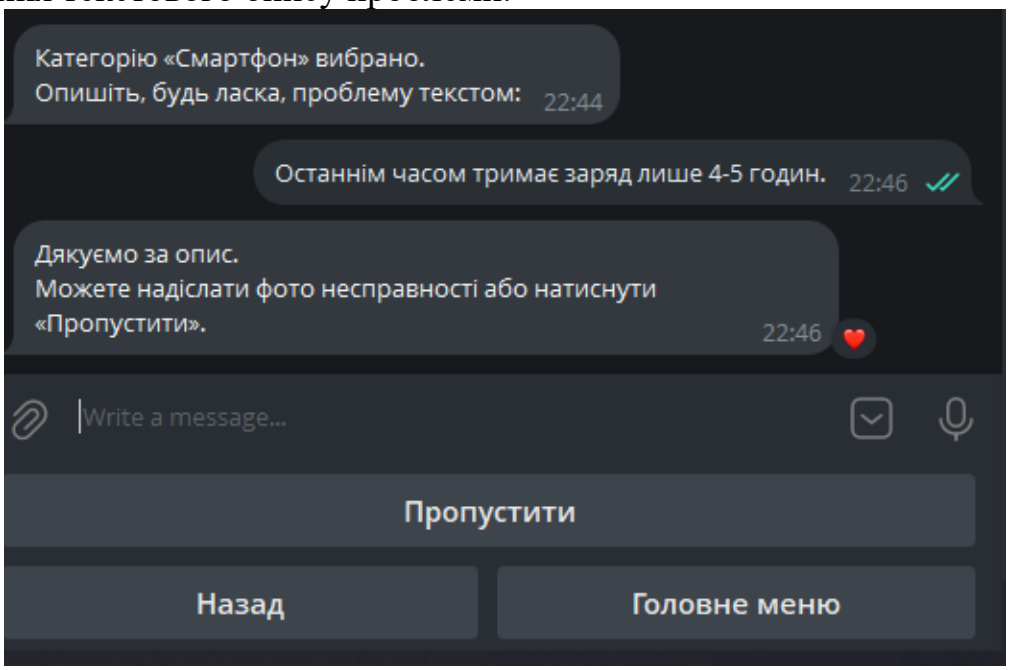


Рисунок 3.31 – Крок 3

Надсилання фотографії несправності.

Введення дати та часу візиту у форматі ДД.ММ.РРРР ГГ:ХХ.

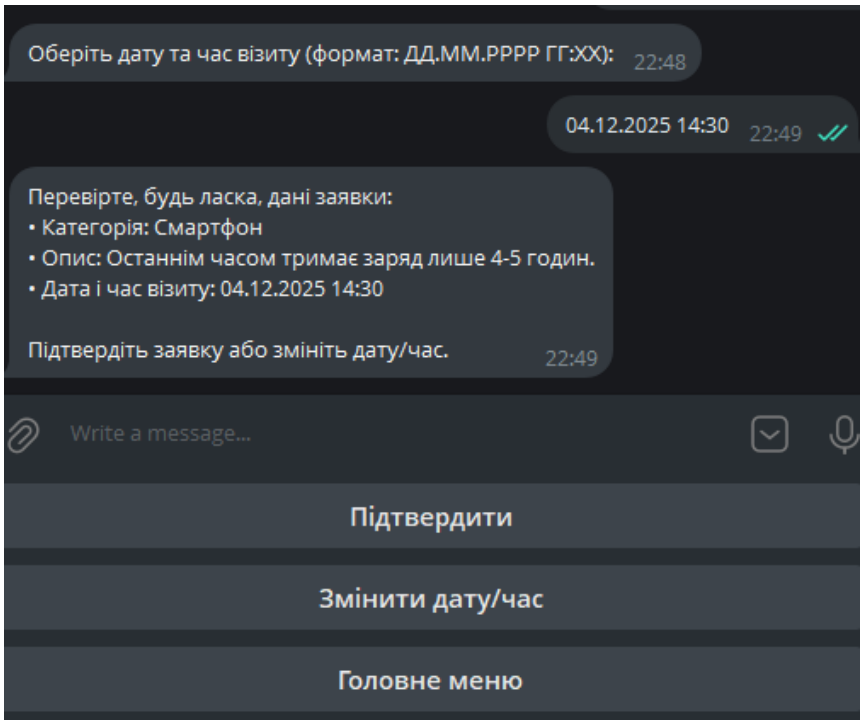


Рисунок 3.32 – Крок 4

Підтвердження заявки кнопкою «Підтвердити».

Очікуваний результат: бот надсилає повідомлення про успішне створення заявки з унікальним номером, дані зберігаються в таблиці applications, статус — «Створена».

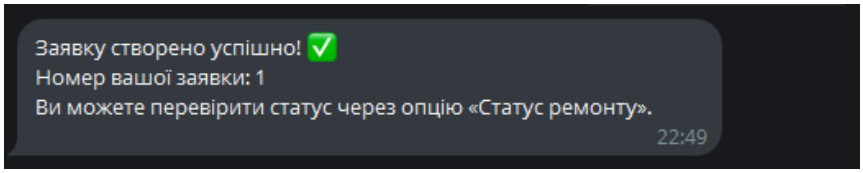


Рисунок 3.33 – Крок 4

Сценарій створення заявки без фото

Аналогічні кроки до етапу опису проблеми.

На етапі завантаження зображення натискається кнопка «Пропустити».

Далі — введення дати та часу й підтвердження.

Очікуваний результат: заявка створюється, поле photo_file_id у БД залишається порожнім, інші дані збережені коректно.

Сценарій перевірки статусу заявки

У головному меню вибирається опція «Статус ремонту».

Користувач вводить номер раніше створеної заявки.

Очікуваний результат: бот повертає інформацію про категорію, заплановану дату візиту та поточний статус ремонту. У разі введення неіснуючого номера — повідомлення про відсутність заявки.

Сценарії перегляду інформаційних розділів

Вибір пунктів «FAQ», «Контакти», «Ціни» в головному меню.

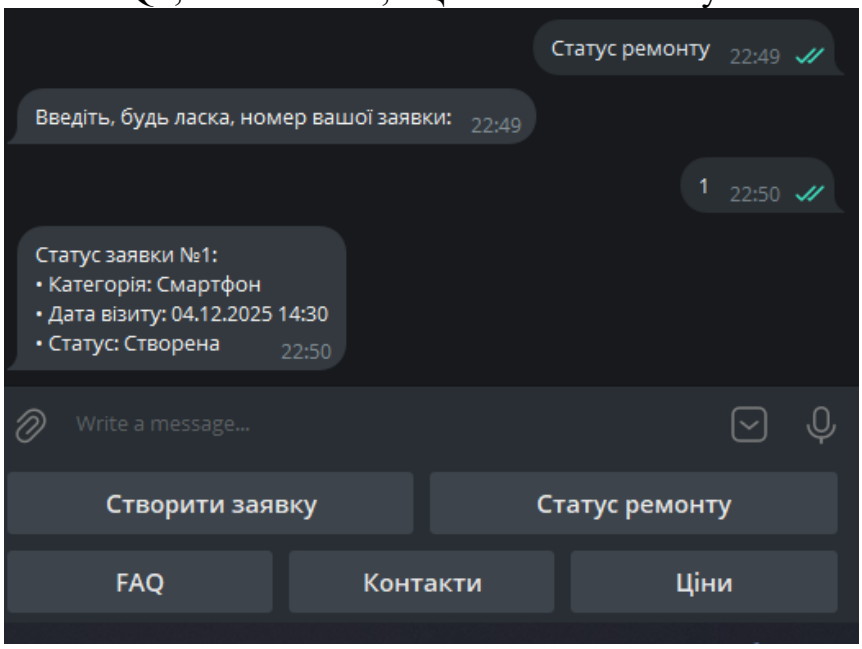


Рисунок 3.34 – Перевірка статусу ремонту

Очікуваний результат: чат-бот виводить відповідні інформаційні блоки та пропонує повернутися до головного меню.

3.5.3 Перевірка оброблення помилок користувача

Окремо було проведено серію тестів, спрямованих на оцінку поведінки системи при некоректних діях користувача:

Некоректний формат дати. Замість коректного формату 25.12.2025 14:30 вводилися варіанти 25-12-25, завтра о 5, 2025/12/25. У всіх випадках бот повертав повідомлення з прикладом правильного формату та залишався в стані вибору дати (S5).

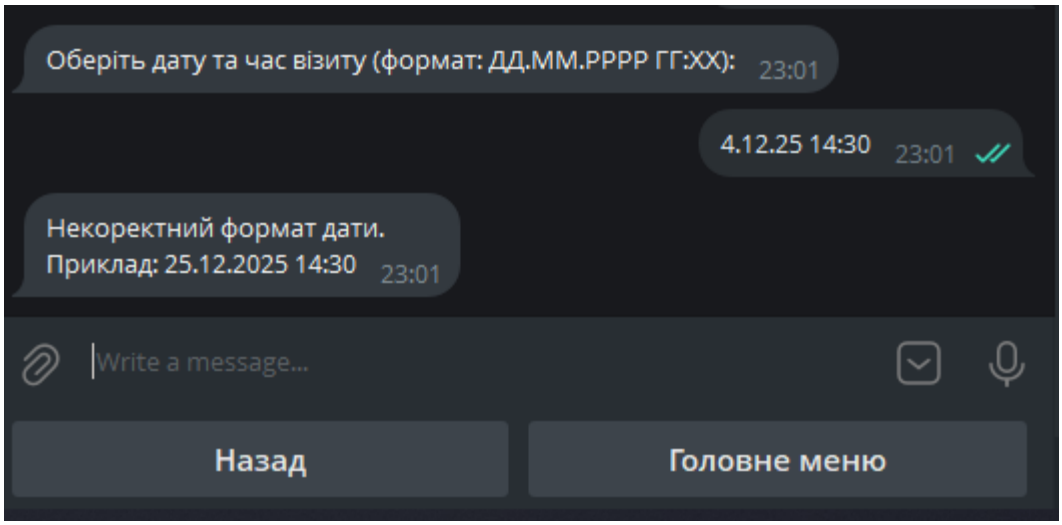


Рисунок 3.35 – Некоректний форат дати

Невірний номер заявки. При введенні випадкового числа, що не відповідає жодній заявці, бот коректно повідомляв про відсутність заявки з таким номером і повертав користувача до головного меню.

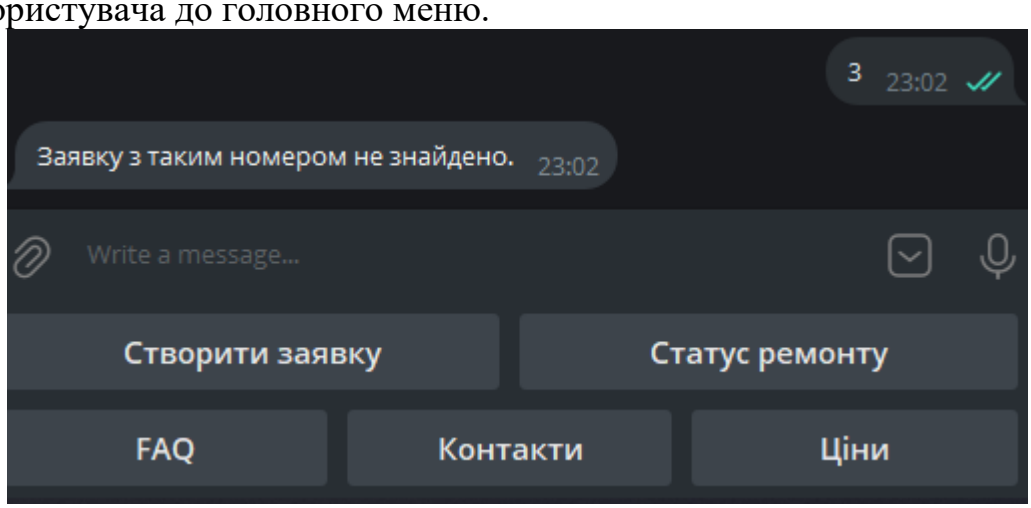


Рисунок 3.36 – невірний номер заявки

Надсилання фото в непередбаченому стані. Якщо користувач надсилав зображення поза станом S4, бот інформував, що фото можна додавати лише на етапі оформлення заявки, і пропонував скористатися головним меню.

Невідомі команди та довільний текст. У режимі головного меню бот ігнорував довільні повідомлення, що не відповідають жодній із кнопок, та пропонував повторно обрати опцію з клавіатури.

Під час тестування не було виявлено критичних помилок, які призводили б до аварійного завершення роботи бота або некоректного збереження даних у БД.

3.5.4 Аналіз результатів та оцінка ефективності

За результатами тестування можна зробити висновок, що розроблений чат-бот:

реалізує всі передбачені сценарії взаємодії відповідно до моделі скінченного автомата;

коректно працює з базою даних: заявки додаються, зчитуються та ідентифікуються за номером;

стійкий до типових помилок користувача, забезпечуючи дружній діалог та підказки щодо виправлення неправильного введення;

забезпечує час створення заявки на рівні близько 1,5 хвилини, що суттєво менше, ніж середня тривалість телефонного звернення до оператора (3–5 хвилин).

Отримані результати підтверджують працездатність розробленої системи та дозволяють зробити висновок про доцільність використання чат-бота як інструмента автоматизації клієнтської підтримки у сервісному центрі з ремонту техніки.

3.6. Реалізація чат-боту через онлайн конструктор

У межах дослідження методів автоматизації клієнтської підтримки доцільно розглянути не лише програмну реалізацію чат-ботів засобами мов програмування, але й альтернативні способи створення подібних систем за допомогою спеціалізованих онлайн конструкторів. Одним із таких інструментів, що набув широкого поширення в Україні та світі, є платформа **BotHelp**, яка дозволяє будувати діалогові сценарії, збирати заявки, виконувати обробку даних та реалізовувати логіку роботи бота без написання програмного коду.

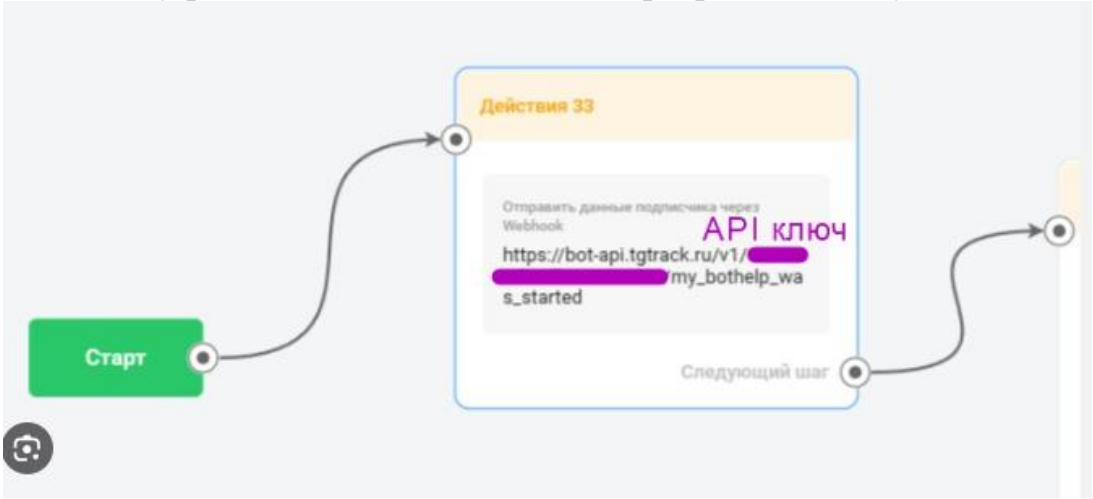


Рисунок 3.37 – Структура роботи BotHelp

Використання онлайн конструкторів є актуальним у випадках, коли розробнику необхідно швидко створити робочий прототип чат-бота, протестувати бізнес-логіку або впровадити базову автоматизацію без залучення програмістів. На відміну від традиційного підходу, що базується на розробці коду мовою Python, конструктори забезпечують візуальне налаштування блоків, сценаріїв та інтеграцій, знижуючи поріг входу та прискорюючи розгортання рішення. Водночас вони мають свої обмеження у гнучкості, масштабованості та можливості створення унікальної логіки, що робить їх додатковим, але не універсальним інструментом.

У цьому підрозділі буде продемонстровано процес створення чат-бота, функціонально ідентичного програмній реалізації, але побудованого з використанням конструктора BotHelp. Наведено структуру сценаріїв, особливості збереження даних, побудови діалогу, а також ключові відмінності між обома підходами. Це дозволить оцінити переваги та недоліки кожного методу з позиції автоматизації клієнтської підтримки та обрати оптимальний інструмент для подальшого використання[13].

3.6.1 Реалізація логіки чат-боту у BotHelp

Для відтворення функціональності чат-бота, реалізованого засобами Python, було створено альтернативну версію бота на основі онлайн конструктора **BotHelp**.

Даний інструмент дозволяє формувати діалогові сценарії у візуальному середовищі, використовуючи блоки, кнопки, змінні та інтеграції, що значно спрощує процес розробки та адаптації логіки без залучення програмування. У цьому підрозділі наведено покроковий процес побудови чат-бота, повністю еквівалентного функціональній структурі програмного рішення.

Рисунок 3.37 – Створення стартового блоку

1. Створення бота та базової конфігурації

Першим етапом було створення нового бота на платформі BotHelp та підключення його до Telegram через токен, отриманий у сервісі BotFather. Після прив’язки чат-бота відкривається доступ до редактора сценаріїв, де кожна частина логіки реалізується у вигляді окремого блоку.

У стартовому блоці формується початкове привітання та створюється головне меню з кнопками:

«Створити заявку»
«Статус ремонту»
«FAQ»
«Контакти»
«Ціни»

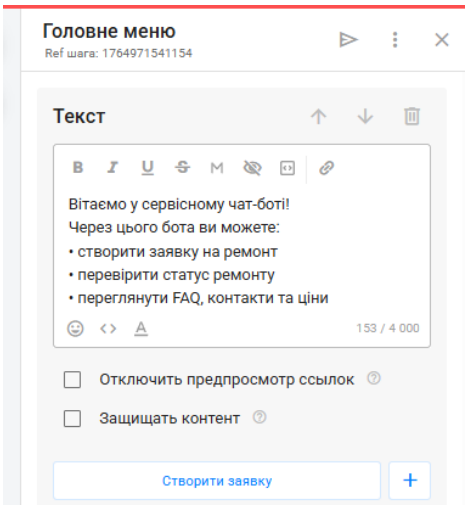


Рисунок 3.38 – Створення стартового блоку

Ці елементи відтворюють головний стан S1, який у Python-версії відповідав за стартову взаємодію з користувачем.

2. Реалізація сценарію створення заявки

Логіка створення заявки в BotHelp побудована за принципом каскадних блоків, які відповідають послідовним станам S2–S6 у Python-боті.

2.1. Вибір категорії техніки

Для реалізації стану S2 створено блок, що містить текстове повідомлення з пропозицією вибрати категорію техніки. Кожна кнопка («Смартфон», «Ноутбук», «ПК», «Побутова техніка») не лише переводить користувача до наступного блоку, але й записує відповідне значення у змінну:

category = "Смартфон"

Таким чином у BotHelp формується еквівалент змінної user_data["category"] із програмної реалізації.

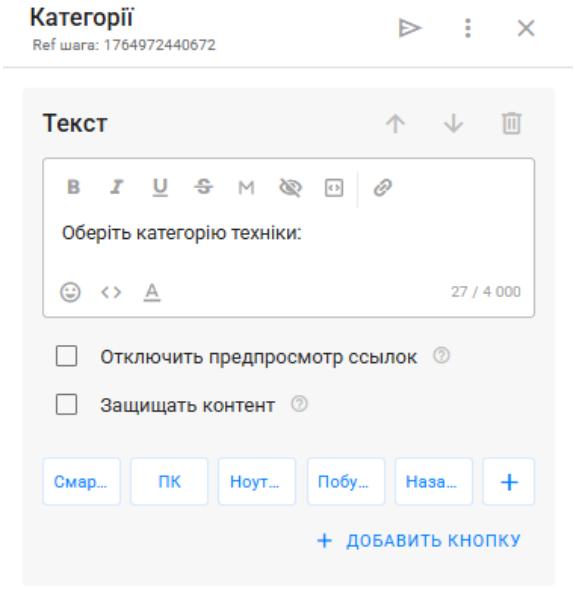


Рисунок 3.39 – Блок категорії

2.2. Введення опису проблеми

Наступний блок реалізує стан S3, де користувач вводить текстовий опис несправності. BotHelp дозволяє увімкнути режим «Очікувати текст» та зберегти введені дані у змінну description. Ці дані використовуються під час формування заявки та у блоці підтвердження.

2.3. Надсилання фото (необов’язково)

Для відтворення стану S4 було створено блок, у якому бот пропонує користувачеві надіслати зображення або натиснути кнопку «Пропустити». У BotHelp передбачено збереження фото у змінну photo, що відповідає ідентифікатору фото в Telegram.

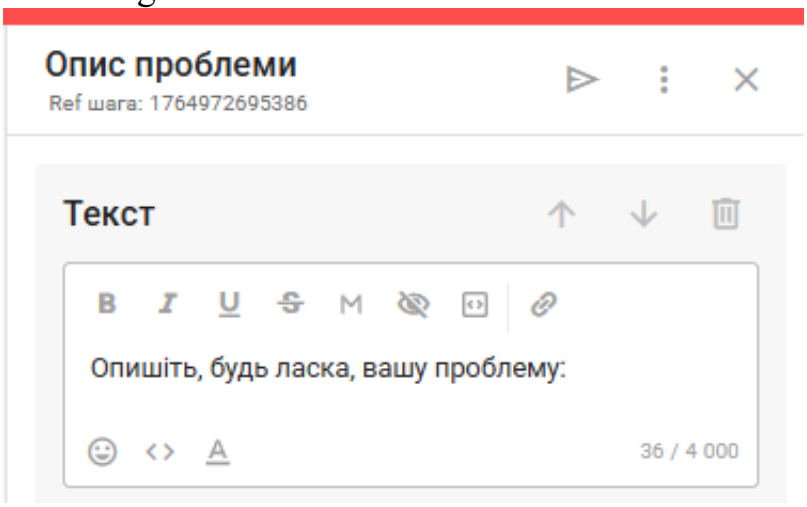


Рисунок 3.40 – Опис проблеми

2.4. Вибір дати і часу візиту

Аналог стану S5 реалізовано шляхом створення блоку з очікуванням текстового введення. Користувач вводить дату та час у довільному форматі, після чого значення зберігається у змінній visit_date. Хоча BotHelp не підтримує автоматичної валідації формату дат, це не заважає функціональній роботі сценарію.

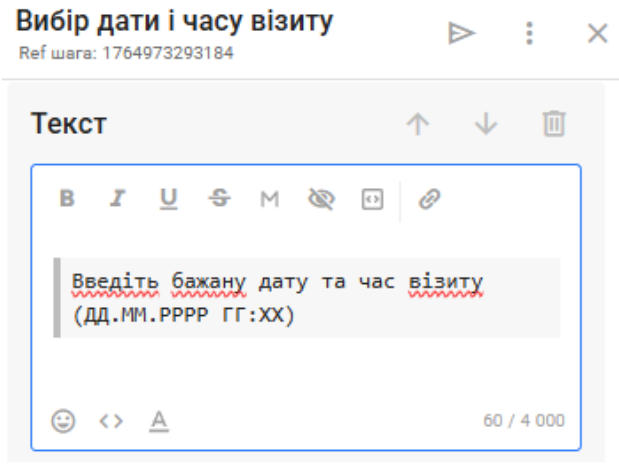


Рисунок 3.41 – Вибір дати і часу візиту

2.5. Підтвердження заявки

Для стану S6 створюється інформаційний блок, який формує попередній перегляд заявки за допомогою змінних:

Категорія: {{category}}
Опис: {{description}}
Дата і час візиту: {{visit_date}}
Користувач може:
підтвердити заявку;
повернутися до зміни дати;
повернутися у головне меню.

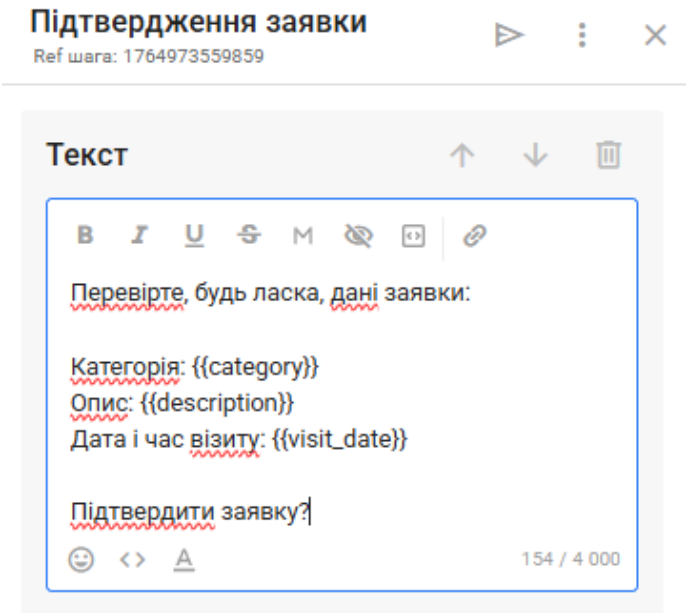


Рисунок 3.42 – Підтвердження заявки

Цей блок повністю дублює логіку підтвердження у Python-боті.

3. Зберігання та обробка заявок

На відміну від Python-реалізації, де дані зберігались у локальній базі SQLite, BotHelp використовує сторонні інтеграції. Найзручнішим варіантом є Google Sheets, яка виступає в ролі простої бази даних.

При підтвердженні заявки запускається дія «Додати рядок», де у таблицю записуються такі поля:

ID (унікальний ідентифікатор звернення у BotHelp)
username користувача
category
description
photo
visit_date
created_at
status = "Створена"

Ця структура повністю повторює SQL-таблицю з програмного варіанту бота.

4. Реалізація механізму перевірки статусу ремонту

Для відтворення стану S7 створюється блок, у якому користувач вводить номер заявки. Далі BotHelp виконує дію «Пошук рядка в Google Sheets», знаходячи відповідний ID.

У разі успіху користувач отримує відповідь формату:

```
Статус заявки №{{app_id}}:
Категорія: {{row.Category}}
Дата візиту: {{row.Visit date}}
Статус: {{row.Status}}
```

Рисунок 3.43 – Підтвердження заявки

При відсутності запису бот видає повідомлення про помилку.

5. Реалізація довідкових блоків

Станам S8–S10 (FAQ, контакти, ціни) відповідають окремі статичні блоки з текстовою інформацією та кнопкою повернення у головне меню. Ці блоки не потребують змінних або складної логіки та повністю повторюють функціональність програмної версії чат-бота.

3.6.2 Тестування

Тестування чат-бота є важливим етапом перевірки коректності роботи діалогових сценаріїв, обробки введених даних та відповідності фактичної поведінки бота очікуваній логіці. У рамках даного дослідження було проведено комплексне тестування чат-бота, створеного на платформі BotHelp, з метою оцінки правильності реалізації основних функцій та виявлення можливих помилок або недоліків.

1. Перевірка коректності діалогових сценаріїв

На першому етапі було здійснено тестування переходів між блоками, які відтворюють логіку станів S1–S10 програмної версії. Перевірка підтвердила:

правильний перехід між основними пунктами меню;

відсутність «розривів» у сценаріях;

коректне повернення до головного меню через спеціальну кнопку;

стабільну роботу гілок «FAQ», «Контакти» та «Ціни».

Усі блоки були пов’язані коректно, і чат-бот у кожному випадку виконував очікувані дії.

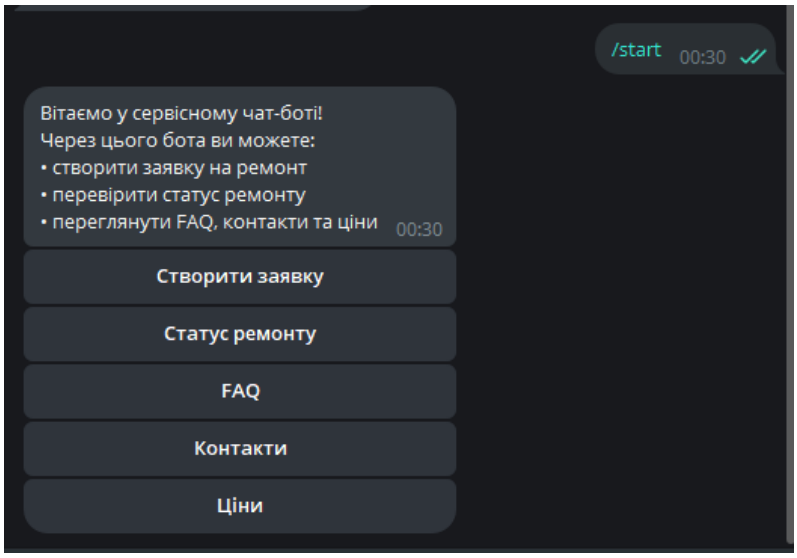


Рисунок 3.44 – Один з етапів тестування

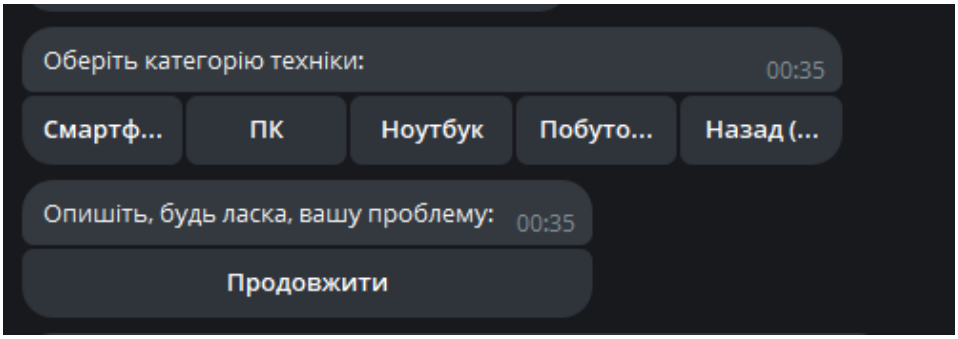


Рисунок 3.45 – Один з етапів тестування

2. Тестування збирання та збереження даних

Особлива увага приділялася перевірці роботи змінних:

category

description

photo

visit_date

У процесі тестування було підтверджено:

дані коректно записуються після кожного кроку;

фото передаються й зберігаються у вигляді Telegram file ID;

дати вводяться у текстовому форматі та коректно відображаються у блоці підтвердження заявки.

Після підтвердження заявка коректно фіксується у Google Sheets, що підтверджує правильність інтеграції.

3. Тестування перевірки статусу заявки

Було перевірено роботу сценарію пошуку заявки за номером. Тестування включало:

введення існуючого номера;

введення неіснуючого номера;
введення некоректних даних.
У всіх тестових випадках бот:
коректно знаходив записи у Google Sheets;
коректно повертав інформацію щодо статусу;
виводив повідомлення про помилку при некоректному номері заявки.
4. Тестування помилкових та нестандартних сценаріїв
Було проведено тестування поведінки бота при:
пропуску окремих полів;
відправленні даних у невідповідному форматі;
повторному натисканні кнопок;
поверненні до попередніх блоків.

Бот стабільно реагував на всі сценарії, не допускаючи аварійних завершень чи логічних помилок.

3.6.3 Аналіз результатів

У результаті проведеної роботи було успішно реалізовано повнофункціональний чат-бот сервісного центру на базі онлайн конструктора BotHelp, який повністю відтворює логіку початкової програмної реалізації на Python. Використання BotHelp дозволило отримати аналогічне рішення з нижчим рівнем технічної складності, високою швидкістю розробки та можливістю централізованого керування діалогами без потреби втручання у програмний код.

Розроблений бот забезпечує:
покрокове оформлення заявки із збиранням усіх необхідних даних;
можливість додавання фото для уточнення проблеми;
формування унікального номера звернення;
автоматичне збереження даних у Google Sheets;
перевірку статусу заявки за її ідентифікатором;
доступ до довідкової інформації (FAQ, контакти, ціни).

Проведене тестування підтвердило коректність роботи всіх елементів чат-бота, стабільність обробки даних та відповідність побудованих сценаріїв вимогам функціональності. Усі ключові бізнес-процеси — від взаємодії з користувачем до формування заявки — працюють передбачено та послідовно.

Таким чином, реалізація чат-бота через онлайн конструктор BotHelp демонструє ефективність використання сучасних інструментів no-code у задачах автоматизації клієнтської підтримки. Порівняння з програмним підходом підтверджує, що конструктори можуть слугувати повноцінною альтернативою для швидкого створення діалогових систем, особливо у випадках, коли немає необхідності у глибокій кастомізації та складній бізнес-логіці.

3.7. Порівняння методів створення чат-ботів

У процесі розробки чат-ботів для автоматизації клієнтської підтримки можуть застосовуватися різні технологічні підходи, кожен з яких має власні переваги, обмеження та сферу доцільного використання. У попередніх підрозділах було детально розглянуто два варіанти реалізації чат-бота: програмний підхід на основі мови Python та використання онлайн конструктора BotHelp, що належить до класу no-code рішень. Хоча обидва підходи дозволяють отримати чат-бот з однаковою функціональністю, принципи їх роботи, вимоги до ресурсів та можливості масштабування суттєво відрізняються.

Тому важливим етапом дослідження є порівняння цих підходів за ключовими критеріями — швидкістю розробки, гнучкістю логіки, вартістю, складністю впровадження, можливостями інтеграції та рівнем технічного контролю. Це дозволить визначити оптимальний спосіб створення чат-ботів залежно від поставлених задач, обсягу проєкту та вимог замовника. Порівняння також допомагає встановити, у яких випадках доцільно застосовувати програмні методи, а коли використання конструкторів є більш ефективним та економічно обґрунтованим.

Таблиця 3.1 – Порівняльна характериистика

Критерій порівняння	Реалізація на Python	Реалізація у BotHelp
Швидкість розробки	Порівняно низька. Необхідно писати код, налаштовувати середовище, БД, хостинг.	Дуже висока. Створення бота від кількох годин до 1 дня завдяки візуальному конструктору.
Необхідні технічні навички	Високі: Python, Telegram API, робота з БД, сервер, FSM, підключення бібліотек.	Мінімальні. Не потребує програмування; достатньо базових навичок роботи з інтерфейсом.
Гнучкість логіки	Максимальна. Можна реалізувати будь-які алгоритми, інтеграції та структури FSM.	Обмежена можливостями конструктора. Складні сценарії або кастомні алгоритми не завжди можливі.
Масштабованість	Висока. Можливо інтегрувати з будь-якими сервісами, БД, хмарними рішеннями.	Середня. Залежить від функціоналу BotHelp та доступних інтеграцій.
Зберігання даних	Локальна або віддалена база (SQLite, MySQL, PostgreSQL). Повний контроль над даними.	Google Sheets або внутрішнє сховище BotHelp. Обмежена гнучкість у структурі даних.

Продовження таблиці 2.5

Інтеграції з іншими системами	Можна підключити будь-який зовнішній API.	Лише доступні інтеграції (CRM, Google Sheets, платіжні сервіси).
Вартість	Безкоштовно (крім хостингу).	Платні тарифи, особливо для WhatsApp/розсилок. Безкоштовний тариф має суттєві обмеження.
Продуктивність та стабільність	Залежить від сервера та якості коду. Може бути дуже високою.	Висока, але залежить від BotHelp та його інфраструктури.
Онлайн-керування та аналітика	Потрібно створювати окремо (адмін-панель, логування, статистика).	Вбудовані інструменти аналітики, перегляду лідів, журналів взаємодії.
Можливість модифікації	Повна. Код доступний і може бути змінений у будь-який момент.	Часткова. Зміни можливі тільки в межах доступних блоків і функцій.
Підтримка мультимедіа	Повна: фото, документи, файли, відео, аудіо.	Часткова: підтримуються основні медіа, але обмежено їх оброблення.
Контроль над бізнес-логікою	Повний. Розробник керує всіма процесами, станами та алгоритмами.	Частковий. Логіка залежить від набору блоків конструктора.
Придатність для складних проєктів	Ідеально. Підходить для великих систем, складної клієнтської підтримки, NLP, AI, інтеграцій.	Підходить для середніх та малих задач: форми, заявки, FAQ, запис даних.
Придатність для швидких прототипів	Низька – потрібно багато коду.	Максимальна – можна створити MVP за 1–2 години.

Згідно з порівняльною таблицею, обидва підходи мають свої сфери застосування.

Python надає повну свободу розробки, високу гнучкість та технічні можливості, що робить його оптимальним рішенням для масштабних та індивідуальних проєктів.

BotHelp, у свою чергу, демонструє високу швидкість створення бота, простоту роботи та зручність підтримки, що дозволяє ефективно використовувати його у проєктах із типовою бізнес-логікою та вимогами до швидкого впровадження.

Рисунок 3.45 – Один з єтапів тестування

Аналіз діаграми та порівняльної таблиці показує, що обидва підходи — програмна реалізація чат-бота на Python та створення бота за допомогою конструктора BotHelp — здатні забезпечити виконання однакових функціональних вимог, однак відрізняються за своїми технічними характеристиками, складністю впровадження та потенціалом масштабування.

Python демонструє значно вищі показники у категоріях гнучкості логіки, масштабованості, інтеграцій з іншими системами та продуктивності. Це пов’язано з тим, що програмна реалізація забезпечує повний контроль над алгоритмами, структурою даних, підключенням до API та адаптацією системи до унікальних вимог. Такий підхід є оптимальним для великих або спеціалізованих проєктів, де необхідна розширювана архітектура та можливість подальшого модифікування.

З іншого боку, BotHelp наочно переважає за параметрами швидкості розробки, простоти використання та низького порогу входу. Візуальне конструювання сценаріїв дозволяє створити повноцінний чат-бот без програмування, що робить цей підхід придатним для малого бізнесу, швидкого прототипування та впровадження стандартних сценаріїв клієнтської підтримки. Проте обмежена гнучкість конструктора може бути критичною у випадках нестандартної логіки або потреби у глибокій інтеграції з корпоративними системами.

Таким чином, результати порівняння підтверджують, що вибір технології залежить від конкретних завдань проєкту:

Python краще підходить для складних, довгострокових та масштабованих систем;

BotHelp — для швидких, простих та економічно ефективних рішень.

Обидва методи можуть доповнювати одне одного в рамках гібридних рішень, де no-code інструменти застосовуються на початкових етапах, а програмні — для подальшого розвитку і розширення функціоналу.

Висновок до розділу 3

У даному розділі було здійснено комплексну реалізацію системи чат-бота для автоматизації клієнтської підтримки сервісного центру двома різними підходами: шляхом створення програмної версії на мові Python та через використання онлайн конструктора BotHelp. Обидві реалізації переслідували спільну мету — забезпечити користувача інтуїтивною та зручною взаємодією, що передбачає створення заявки на ремонт, збирання та структуризацію інформації, а також можливість перегляду статусу звернення.

Реалізація чат-бота засобами Python продемонструвала високу гнучкість та можливість побудови індивідуальної логіки завдяки використанню механізму кінцевого автомата станів (FSM), роботи з локальною базою даних SQLite та прямої взаємодії з Telegram API. Такий підхід забезпечує повний контроль над внутрішньою архітектурою, дозволяє масштабувати систему, інтегрувати додаткові модулі, розширювати функціональність та адаптувати бота під специфічні бізнес-вимоги. Програмна версія є найбільш придатною у випадках, коли потрібна висока кастомізація, складні алгоритми обробки даних або інтеграція з зовнішніми інформаційними системами.

У свою чергу, використання конструктора BotHelp показало переваги безкодових технологій (no-code), які дозволяють значно швидше створювати робочі прототипи та впроваджувати повноцінні чат-боти. BotHelp надав інструменти для візуального конструювання сценаріїв, зберігання даних, автоматизації дій та інтеграцій, зокрема з Google Sheets. Відтворення логіки Python-версії у BotHelp підтвердило, що конструктори здатні ефективно вирішувати типові бізнес-задачі, пов’язані зі збором заявок, маршрутизацією діалогів та формуванням відповідей без необхідності залучення розробників.

Порівняльний аналіз двох підходів показав, що **Python-реалізація забезпечує максимальну функціональну гнучкість**, тоді як **BotHelp — швидкість розробки, простоту використання та доступність для користувачів без технічного досвіду**. У практичних умовах вибір між двома методами залежить від цілей проєкту: для комплексних систем та унікальних алгоритмів доцільні програмні рішення, а для стандартних процесів обробки заявок та швидкого розгортання — no-code конструктори.

Таким чином, проведена реалізація обох підходів довела, що сучасні інструменти розробки чат-ботів можуть ефективно взаємодоповнювати один одного: програмний підхід забезпечує архітектурну свободу, тоді як конструктор забезпечує швидкість і простоту. Обидва рішення виконують поставлені функціональні вимоги, демонструючи можливість широкого використання чат-ботів для підвищення якості та ефективності клієнтської підтримки в сервісних центрах.

ВИСНОВОК

У магістерській кваліфікаційній роботі було комплексно досліджено теоретичні засади, методологічні принципи та практичні підходи до створення систем автоматизації клієнтської підтримки на основі чат-ботів. Проведене дослідження охоплює широкий спектр питань — від аналізу сучасного стану цифрових сервісів і технологій обробки природної мови до проєктування, реалізації та тестування автоматизованої системи підтримки користувачів, інтегрованої з популярними комунікаційними каналами.

У першому розділі здійснено фундаментальний аналіз теоретичних аспектів проблематики. Встановлено, що традиційні системи клієнтської підтримки, засновані на роботі операторів, мають низку суттєвих недоліків: обмеженість швидкості реагування, залежність від людського фактора, високі витрати часу та ресурсів, а також неможливість забезпечити безперервну доступність для користувачів. Проведений огляд наукових джерел підтвердив, що автоматизація цих процесів є одним із ключових напрямів розвитку сучасних інформаційних технологій.

Особливу увагу приділено еволюції чат-ботів — від елементарних сценарних моделей до інтелектуальних систем, що використовують методи NLP, машинного навчання та аналізу контексту. Розглянуто класифікацію ботів, специфіку їх архітектури, принципи побудови діалогових систем, а також можливості існуючих фреймворків і платформ. Визначено, що чат-боти є ефективним інструментом оптимізації бізнес-процесів, оскільки забезпечують зменшення навантаження на персонал, підвищення швидкості обробки запитів і покращення якості обслуговування.

У другому розділі детально опрацьовано технічні аспекти проєктування системи. Порівняння сучасних мов програмування показало, що Python є оптимальним вибором завдяки своїй універсальності, широкій екосистемі бібліотек та потужним інструментам для NLP і API-інтеграцій. Розроблено модульну архітектуру системи, що включає підсистему обробки запитів, NLP-модуль, базу знань, механізми роботи з даними та модулі інтеграції з Telegram і зовнішніми сервісами.

Алгоритм роботи чат-бота охоплює повний цикл обробки звернення: отримання повідомлення, його попередню фільтрацію, визначення інтенції, пошук відповідної відповіді у базі знань або запит до мовної моделі, формування відповіді та її передачу користувачу. Створення формальної моделі кінцевого автомата (11 станів, 13 команд, 23 переходи) забезпечило структурованість

діалогу та можливість точного керування сценаріями. Важливою особливістю запропонованої архітектури є її здатність до самонавчання на основі аналізу історії діалогів, що створює передумови для довготривалого розвитку системи, підвищення точності відповідей і адаптивності до потреб користувачів.

Практична частина роботи присвячена реалізації двох версій чат-бота — програмної (Python) та no-code (BotHelp). Python-версія продемонструвала гнучкість, глибоку кастомізацію та повний контроль над логікою, станами, даними та алгоритмами. Її архітектура дозволяє масштабувати систему, інтегрувати додаткові функції та застосовувати інтелектуальні методи аналізу текстів. Реалізація чат-бота на платформі BotHelp показала інший підхід: швидке створення сценаріїв, зручна робота з візуальними блоками, легкість у модифікації та можливість побудови автоматизованих процесів без програмування. Створений бот повністю відтворив логіку Python-версії, включно зі збиранням заявок, обробкою даних та перевіркою їх статусу.

Порівняльний аналіз двох підходів засвідчив, що:

Python — оптимальний для складних, гнучких, інтелектуальних та масштабованих систем;

BotHelp — ефективний для швидкого прототипування, впровадження стандартних сценаріїв і систем із мінімальними вимогами до кастомізації.

Тестування обох реалізацій підтвердило стабільність роботи, коректність алгоритмів і можливість практичного впровадження в реальних умовах. Обидва рішення забезпечують можливість оформлення заявок, їх збереження, обробки та перевірки статусу, а також надання довідкової інформації.

У підсумку можна стверджувати, що методи автоматизації клієнтської підтримки на основі чат-ботів є надзвичайно перспективними для впровадження в сервісних центрах, корпоративних структурах, освітніх установах і багатьох інших сферах. Розроблені рішення демонструють, що такі системи здатні забезпечувати безперервний доступ до підтримки, підвищувати швидкість реагування, зменшувати операційні витрати та покращувати загальний рівень взаємодії з користувачами. Поєднання теоретичних, технічних та практичних результатів роботи створює міцний фундамент для подальшого розвитку системи, інтеграції нових функцій, застосування моделей штучного інтелекту та розширення можливостей автоматизованої взаємодії.

Виконана кваліфікаційна робота підтверджує, що чат-боти є важливим напрямом цифрової трансформації бізнес-процесів і мають значний потенціал для подальшого впровадження, удосконалення та масштабування.

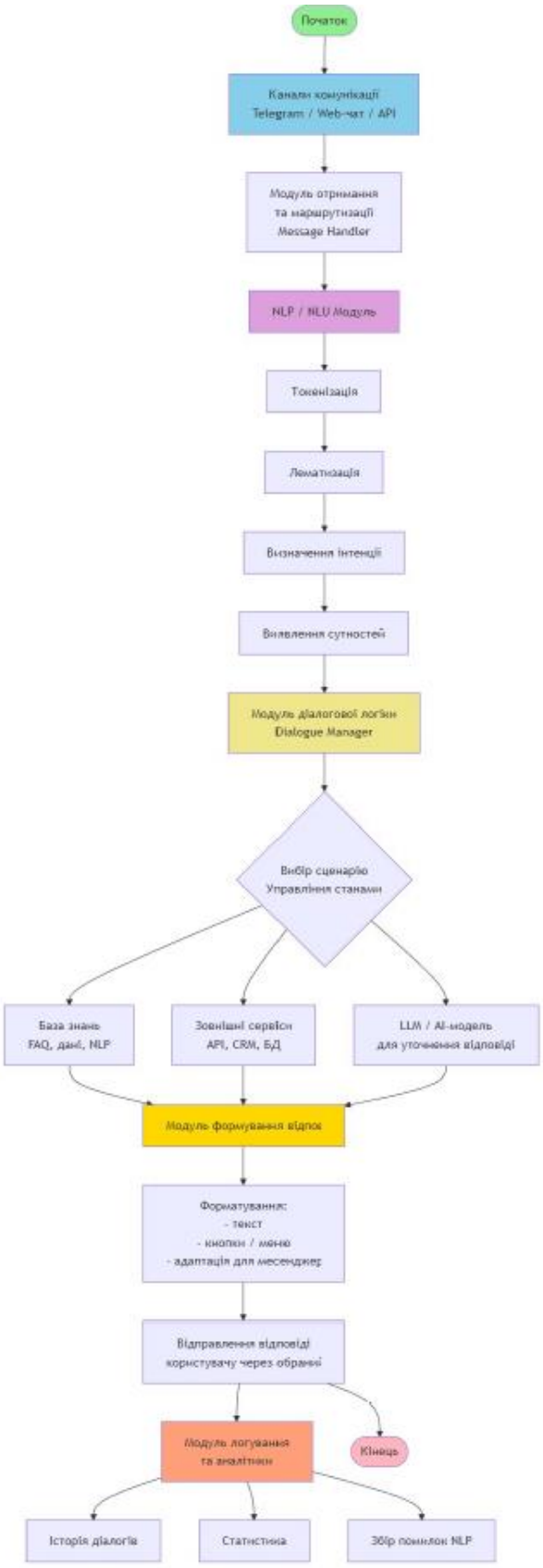
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adamopoulou, E., & Moussiades, L. *An overview of chatbot technology*. Proceedings of the 16th Artificial Intelligence Applications and Innovations Conference, 2020.
2. Følstad, A., & Brandtzæg, P. B. *Chatbots and the new world of HCI*. Interactions, 24(4), 38–42, 2017.
3. Jain, M., Kumar, P., Kota, R., & Patel, S. N. *Evaluating and Informing the Design of Chatbots*. CHI ’18 Proceedings, ACM, 2018.
4. Radziwill, N., & Benton, M. *Evaluating Quality of Chatbots and Intelligent Conversational Agents*. Journal of Intelligent Systems, 2017.
5. Dale, R. The return of the chatbots. Natural Language Engineering, Cambridge University Press, 2016.
6. Chung, M., Ko, E., Joung, H., & Kim, S. J. *Chatbot e-service and customer satisfaction*. Journal of Retailing and Consumer Services, 56, 2020.
7. Terence, R., et al. *A Survey on NLP for Chatbot Systems: Architectures, Approaches, and Applications*. IEEE Access, 2021.
8. Jurafsky, D., & Martin, J. H. *Speech and Language Processing*. 3rd edition draft, 2023.
9. Bird, S., Klein, E., & Loper, E. *Natural Language Processing with Python*. O’Reilly Media, 2009.
10. Russell, S., & Norvig, P. *Artificial Intelligence: A Modern Approach*. 4th Edition, Pearson, 2021.
11. López, G., Quesada, L., & Guerrero, L. A. *Chatbots: History, technology, and applications*. Springer, 2021.
12. Telegram Bot API — Official Documentation. https://core.telegram.org/bots/api
13. Python Software Foundation. Official Python Documentation. https://docs.python.org/
14. SQLite Consortium. SQLite documentation. https://www.sqlite.org/docs.html
15. TeleBot / pyTelegramBotAPI documentation (GitHub).
16. Ammon, S. Building Chatbots with Python. Apress, 2020.
17. Tredinnick, L. Python for Data and Text Analysis. CRC Press, 2018.
18. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *NAACL*, 2019.

19. Brown, T. et al. GPT-3: Language Models are Few-Shot Learners. *NeurIPS*, 2020.
20. Hirschberg, J., & Manning, C. *Natural Language Processing and AI*. Communications of the ACM, 2015.
21. Meyer, C., Schwager, A. Understanding Customer Experience**.** *Harvard Business Review*, 2007.
22. Payne, A., & Frow, P. *A Strategic Framework for Customer Relationship Management.* Journal of Marketing, 2005.
23. Chaffey, D. *Digital Business and E-Commerce Management***.** Pearson, 2019.

ДОДАТОК А

Блок-схема алгоритму роботи чат-бота



ДОДАТОК Б

Код чат-бота

```
import logging
import sqlite3
from datetime import datetime

import telebot
from telebot import types

BOT_TOKEN = "TELEGRAM_BOT_TOKEN "
DB_NAME = "applications.db"

bot = telebot.TeleBot(BOT_TOKEN)

logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s",
    level=logging.INFO
)
logger = logging.getLogger(__name__)

class States:
    S0 = "S0"   # Початковий
    S1 = "S1"   # Головне меню
    S2 = "S2"   # Вибір категорії
    S3 = "S3"   # Опис проблеми
    S4 = "S4"   # Завантаження фото
    S5 = "S5"   # Вибір дати/часу
    S6 = "S6"   # Підтвердження
    S7 = "S7"   # Перевірка статусу
    S8 = "S8"   # FAQ
    S9 = "S9"   # Контакти
    S10 = "S10" # Прайс

user_states: dict[int, str] = {}
user_data: dict[int, dict] = {}


def get_state(user_id: int) -> str:
    return user_states.get(user_id, States.S1)
```

```
def set_state(user_id: int, state: str):
    user_states[user_id] = state
    # гарантуємо наявність словника даних для користувача
    user_data.setdefault(user_id, {})

def init_db():
    conn = sqlite3.connect(DB_NAME)
    cur = conn.cursor()
    cur.execute(
        """
        CREATE TABLE IF NOT EXISTS applications (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER,
            username TEXT,
            category TEXT,
            description TEXT,
            photo_file_id TEXT,
            visit_datetime TEXT,
            status TEXT DEFAULT 'Створена',
            created_at TEXT
        )
        """
    )
    conn.commit()
    conn.close()

def main_menu_keyboard():
    kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
    kb.row(types.KeyboardButton("Створити заявку"),
           types.KeyboardButton("Статус ремонту"))
    kb.row(types.KeyboardButton("FAQ"),
           types.KeyboardButton("Контакти"),
           types.KeyboardButton("Ціни"))
    return kb


def back_keyboard():
    kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
    kb.row(types.KeyboardButton("Назад"),
           types.KeyboardButton("Головне меню"))
    return kb
```

```
def skip_or_back_keyboard():
    kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
    kb.row(types.KeyboardButton("Пропустити"))
    kb.row(types.KeyboardButton("Назад"),
           types.KeyboardButton("Головне меню"))
    return kb

def confirm_keyboard():
    kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
    kb.row(types.KeyboardButton("Підтвердити"))
    kb.row(types.KeyboardButton("Змінити дату/час"))
    kb.row(types.KeyboardButton("Головне меню"))
    return kb

def categories_keyboard():
    kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
    kb.row(types.KeyboardButton("Смартфон"),
           types.KeyboardButton("Ноутбук"))
    kb.row(types.KeyboardButton("ПК"),
           types.KeyboardButton("Побутова техніка"))
    kb.row(types.KeyboardButton("Назад"),
           types.KeyboardButton("Головне меню"))
    return kb

# ===== /start =====
@bot.message_handler(commands=["start"])
def cmd_start(message: types.Message):
    user_id = message.from_user.id
    set_state(user_id, States.S1)
    bot.send_message(
        message.chat.id,
        "Вітаємо у сервісному чат-боті!\n"
        "Через цього бота ви можете створити заявку на ремонт, "
        "перевірити статус, переглянути FAQ, контакти та ціни.",
        reply_markup=main_menu_keyboard(),
    )

@bot.message_handler(content_types=["text"])
def handle_text(message: types.Message):
    text = message.text.strip()
```

```
user = message.from_user
user_id = user.id
state = get_state(user_id)

# Глобальна команда "Головне меню"
if text.lower() == "головне меню":
    set_state(user_id, States.S1)
    bot.send_message(
        message.chat.id,
        "Повернення до головного меню.",
        reply_markup=main_menu_keyboard(),
    )
    return

# ---------- S1: Головне меню ----------
if state == States.S1:
    if text == "Створити заявку":
        set_state(user_id, States.S2)
        bot.send_message(
            message.chat.id,
            "Оберіть категорію техніки:",
            reply_markup=categories_keyboard(),
        )
    elif text == "Статус ремонту":
        set_state(user_id, States.S7)
        bot.send_message(
            message.chat.id,
            "Введіть, будь ласка, номер вашої заявки:",
            reply_markup=back_keyboard(),
        )
    elif text == "FAQ":
        set_state(user_id, States.S8)
        bot.send_message(
            message.chat.id,
            "Часті питання:\n"
            "1. Як довго триває ремонт?\n"
            "2. Чи є гарантія на ремонт?\n"
            "3. Чи можна оформити кур’єрську доставку?\n\n"
            "Оберіть «Головне меню» для повернення.",
            reply_markup=types.ReplyKeyboardMarkup(
                resize_keyboard=True
```

```
            ).add(types.KeyboardButton("Головне меню")),
        )
    elif text == "Контакти":
        set_state(user_id, States.S9)
        kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
        kb.add(types.KeyboardButton("Головне меню"))
        bot.send_message(
            message.chat.id,
            "Контакти сервісного центру:\n"
            "🔎 Адреса: м. Київ, вул. Прикладна, 10\n"
            "📞 Телефон: +38 (000) 000-00-00\n"
            "🕒 Графік: Пн–Пт, 9:00–18:00",
            reply_markup=kb,
        )
    elif text == "Ціни":
        set_state(user_id, States.S10)
        kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
        kb.row(types.KeyboardButton("Головне меню"),
               types.KeyboardButton("Створити заявку"))
        bot.send_message(
            message.chat.id,
            "Орієнтовний прайс:\n"
            "• Діагностика — від 200 грн\n"
            "• Ремонт смартфонів — від 500 грн\n"
            "• Ремонт ноутбуків — від 800 грн\n"
            "• Ремонт побутової техніки — від 600 грн",
            reply_markup=kb,
        )
    else:
        bot.send_message(
            message.chat.id,
            "Не вдалося розпізнати команду. Оберіть опцію з меню.",
            reply_markup=main_menu_keyboard(),
        )
    return

# ---------- S2: Вибір категорії ----------
if state == States.S2:
    if text == "Назад":
        set_state(user_id, States.S1)
        bot.send_message(
```

```
            message.chat.id,
            "Повернення до головного меню.",
            reply_markup=main_menu_keyboard(),
        )
        return
    if text in ["Смартфон", "Ноутбук", "ПК", "Побутова техніка"]:
        data = user_data.setdefault(user_id, {})
        data["category"] = text
        set_state(user_id, States.S3)
        bot.send_message(
            message.chat.id,
            f"Категорію «{text}» вибрано.\n"
            "Опишіть, будь ласка, проблему текстом:",
            reply_markup=back_keyboard(),
        )
    else:
        bot.send_message(
            message.chat.id,
            "Будь ласка, оберіть категорію з клавіатури.",
            reply_markup=categories_keyboard(),
        )
    return

# ---------- S3: Опис проблеми ----------
if state == States.S3:
    if text == "Назад":
        set_state(user_id, States.S2)
        bot.send_message(
            message.chat.id,
            "Повертаємось до вибору категорії.",
            reply_markup=categories_keyboard(),
        )
        return
    data = user_data.setdefault(user_id, {})
    data["description"] = text
    set_state(user_id, States.S4)
    bot.send_message(
        message.chat.id,
        "Дякуємо за опис.\n"
        "Можете надіслати фото несправності або натиснути «Пропустити».",
        reply_markup=skip_or_back_keyboard(),
```

```
    )
    return

# ---------- S4: Фото або пропуск ----------
if state == States.S4:
    if text == "Назад":
        set_state(user_id, States.S3)
        bot.send_message(
            message.chat.id,
            "Повернення до введення опису.",
            reply_markup=back_keyboard(),
        )
        return
    if text == "Пропустити":
        data = user_data.setdefault(user_id, {})
        data["photo_file_id"] = None
        set_state(user_id, States.S5)
        bot.send_message(
            message.chat.id,
            "Оберіть дату та час візиту (формат: ДД.ММ.РРРР ГГ:ХХ):",
            reply_markup=back_keyboard(),
        )
        return
    bot.send_message(
        message.chat.id,
        "Якщо ви не хочете надсилати фото, натисніть «Пропустити».",
        reply_markup=skip_or_back_keyboard(),
    )
    return

# ---------- S5: Вибір дати/часу ----------
if state == States.S5:
    if text == "Назад":
        set_state(user_id, States.S4)
        bot.send_message(
            message.chat.id,
            "Повернення до завантаження фото.",
            reply_markup=skip_or_back_keyboard(),
        )
        return
    try:
```

```
            dt = datetime.strptime(text, "%d.%m.%Y %H:%M")
            data = user_data.setdefault(user_id, {})
            data["visit_datetime"] = dt.strftime("%d.%m.%Y %H:%M")
        except ValueError:
            bot.send_message(
                message.chat.id,
                "Некоректний формат дати.\n"
                "Приклад: 25.12.2025 14:30",
                reply_markup=back_keyboard(),
            )
            return

        set_state(user_id, States.S6)
        data = user_data.setdefault(user_id, {})
        category = data.get("category", "-")
        desc = data.get("description", "-")
        dt_str = data.get("visit_datetime", "-")
        bot.send_message(
            message.chat.id,
            "Перевірте, будь ласка, дані заявки:\n"
            f"• Категорія: {category}\n"
            f"• Опис: {desc}\n"
            f"• Дата і час візиту: {dt_str}\n\n"
            "Підтвердіть заявку або змініть дату/час.",
            reply_markup=confirm_keyboard(),
        )
        return

    # ---------- S6: Підтвердження ----------
    if state == States.S6:
        if text == "Підтвердити":
            data = user_data.setdefault(user_id, {})
            category = data.get("category")
            desc = data.get("description")
            photo_id = data.get("photo_file_id")
            dt_str = data.get("visit_datetime")

            conn = sqlite3.connect(DB_NAME)
            cur = conn.cursor()
            cur.execute(
                """
```

```
            INSERT INTO applications
            (user_id, username, category, description, photo_file_id, visit_datetime,
status, created_at)
            VALUES (?, ?, ?, ?, ?, ?, 'Створена', ?)
            """,
            (
                user_id,
                user.username,
                category,
                desc,
                photo_id,
                dt_str,
                datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            ),
        )
        app_id = cur.lastrowid
        conn.commit()
        conn.close()

        set_state(user_id, States.S1)
        bot.send_message(
            message.chat.id,
            f"Заявку створено успішно! ✅\n"
            f"Номер вашої заявки: {app_id}\n"
            "Ви можете перевірити статус через опцію «Статус ремонту».",
            reply_markup=main_menu_keyboard(),
        )
    elif text == "Змінити дату/час":
        set_state(user_id, States.S5)
        bot.send_message(
            message.chat.id,
            "Введіть нову дату та час (формат: ДД.ММ.РРРР ГГ:ХХ):",
            reply_markup=back_keyboard(),
        )
    else:
        bot.send_message(
            message.chat.id,
            "Оберіть «Підтвердити» або «Змінити дату/час».",
            reply_markup=confirm_keyboard(),
        )
    return
```

```
# ---------- S7: Перевірка статусу ----------
if state == States.S7:
    if text == "Назад":
        set_state(user_id, States.S1)
        bot.send_message(
            message.chat.id,
            "Повернення до головного меню.",
            reply_markup=main_menu_keyboard(),
        )
        return
    try:
        app_id = int(text)
    except ValueError:
        bot.send_message(
            message.chat.id,
            "Номер заявки має бути числом. Спробуйте ще раз.",
            reply_markup=back_keyboard(),
        )
        return

    conn = sqlite3.connect(DB_NAME)
    cur = conn.cursor()
    cur.execute(
        "SELECT status, category, visit_datetime FROM applications WHERE id = ?",
        (app_id,),
    )
    row = cur.fetchone()
    conn.close()

    if row:
        status, category, dt_str = row
        bot.send_message(
            message.chat.id,
            f"Статус заявки №{app_id}:\n"
            f"• Категорія: {category}\n"
            f"• Дата візиту: {dt_str}\n"
            f"• Статус: {status}",
            reply_markup=main_menu_keyboard(),
        )
    else:
```

```
            bot.send_message(
                message.chat.id,
                "Заявку з таким номером не знайдено.",
                reply_markup=main_menu_keyboard(),
            )

        set_state(user_id, States.S1)
        return

    # ---------- S8, S9, S10 ----------
    if state in [States.S8, States.S9, States.S10]:
        kb = types.ReplyKeyboardMarkup(resize_keyboard=True)
        kb.add(types.KeyboardButton("Головне меню"))
        bot.send_message(
            message.chat.id,
            "Оберіть «Головне меню» для продовження роботи.",
            reply_markup=kb,
        )
        return

    # Невідомий стан — скидаємо в головне меню
    set_state(user_id, States.S1)
    bot.send_message(
        message.chat.id,
        "Сталася помилка стану. Повертаю вас до головного меню.",
        reply_markup=main_menu_keyboard(),
    )

@bot.message_handler(content_types=["photo"])
def handle_photo(message: types.Message):
    user_id = message.from_user.id
    state = get_state(user_id)

    if state != States.S4:
        bot.send_message(
            message.chat.id,
            "Фото можна надсилати лише на етапі оформлення заявки.",
            reply_markup=main_menu_keyboard(),
        )
        return
```

```
    photo = message.photo[-1]
    file_id = photo.file_id
    data = user_data.setdefault(user_id, {})
    data["photo_file_id"] = file_id

    set_state(user_id, States.S5)
    bot.send_message(
        message.chat.id,
        "Фото збережено. Введіть дату та час візиту (формат: ДД.ММ.РРРР ГГ:ХХ):",
        reply_markup=back_keyboard(),
    )

if __name__ == "__main__":
    init_db()
    print("Бот запущено. Відкрий Telegram і пройди сценарій створення заявки.")
    bot.infinity_polling()
```