ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Telegram-бот для вивчення іноземних слів з використанням JavaScript та Node.js»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерних наук
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

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

_____ *Марія ЛОЗЕНКО*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КНД-42
Марія ЛОЗЕНКО
Ім'я, ПРІЗВИЩЕ

Керівник: Юлія БЕРЕЗОВСЬКА
науковий ступінь, вчене звання: доктор філософії *доценнт*

Рецензент: _____
науковий ступінь, вчене звання

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____ Віктор ВИШНІВСЬКИЙ
_____
«_____» _____ 2025р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Лозенко Марія Сергіївна
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Telegram-бот для вивчення іноземних слів з використанням JavaScript та Node.js
керівник кваліфікаційної роботи Юлія БЕРЕЗОВСЬКА, докторо філософії, доцент,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025р. № 56
2. Строк подання студентом роботи «30» травня 2025р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література; Telegram Bot API; бібліотека Telegraf; мова програмування JavaScript; середовище виконання Node.js; база даних MongoDB; ODM-бібліотека Mongoose; система керування версіями даних; контейнеризація; Docker; інтервальне повторення (Spaced Repetition); .env-змінні; архітектура чат-бота; асинхронне програмування; REST API; обробка повідомлень; інлайн-клавіатура Telegram; сесійне зберігання стану; користувацький інтерфейс (CLI); серіалізація JSON; середовище розгортання.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Аналіз технологій для вивчення іноземних слів
    2. Вибір і опис технологій для розробки бота
    3. Створення telegram-бота для вивчення іноземних слів
5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання «25» лютого 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	25.02-24.03.2025	Виконано
2	Обробка матеріалу	23.03-05.04.2025	Виконано
3	Розробка теоретичної частини	06.04-10.04.2025	Виконано
4	Розробка telegram-бот	11.04-20.04.2025	Виконано
5	Висновки за результатами аналізу	21.04-30.04.2025	Виконано
6	Оформлення пояснювальної записки	01.05-05.05.2025	Виконано
7	Розробка презентації	05.05-10.05.2025	Виконано
8	Передзахист	26.05-28.05.2025	Виконано

Здобувач вищої освіти _____ Марія ЛОЗЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Юлія БЕРЕЗОВСЬКА
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 51 стор., 40 рис., 1 табл., 15 джерел.

Мета роботи – розробка Telegram-бота для вивчення іноземних слів з використанням JavaScript та Node.js.

Об'єкт дослідження – інформаційна система для вивчення іноземної лексики у форматі чат-бота.

Предмет дослідження – telegram-бот як засіб організації процесу вивчення іноземних слів.

У сучасному світі зростає попит на ефективні та зручні інструменти для самостійного вивчення іноземних мов. Одним із таких рішень є чат-боти в месенджерах, які забезпечують швидкий доступ до навчального контенту без потреби у встановленні додаткових програм. Telegram-боти стали популярним вибором завдяки простоті використання, кросплатформеності та широким можливостям інтеграції.

У роботі проаналізовано існуючі підходи до цифрового навчання іноземних мов, розглянуто типи чат-ботів, оцінено переваги та недоліки порівняно з мобільними застосунками. Для реалізації Telegram-бота було обрано стек технологій: JavaScript, Node.js, бібліотеку Telegraf, MongoDB як базу даних, та Docker для контейнеризації. Основний функціонал бота полягає у додаванні слів, інтервальному повторенні, перевірці знань та статистиці навчання.

На основі розробленого рішення продемонстровано ефективність використання Telegram-бота як інструменту для покращення словникового запасу користувача.

КЛЮЧОВІ СЛОВА: TELEGRAM-БОТ, JAVASCRIPT, NODE.JS, MONGODB, DOCKER, TELEGRAF, ІНТЕРВАЛЬНЕ ПОВТОРЕННЯ, ВИВЧЕННЯ ІНОЗЕМНИХ СЛІВ, ЧАТ-БОТ, БОТ ДЛЯ НАВЧАННЯ.

ЗМІСТ

ВСТУП

Сучасні технології активно змінюють способи засвоєння інформації та організації навчального процесу. З появою широкого доступу до Інтернету та зростанням популярності мобільних пристроїв, все більшої актуальності набувають цифрові інструменти для самостійного навчання, зокрема — сервіси для вивчення іноземних мов. Одним із таких зручних та ефективних засобів стали чат-боти, які функціонують у популярних месенджерах.

Telegram-боти вирізняються легкістю доступу, швидкістю відповіді, простотою у використанні та можливістю персоналізованої взаємодії з користувачем. Саме ці якості роблять їх привабливими для освітніх цілей, зокрема для тренування словникового запасу. У порівнянні з традиційними мобільними застосунками, Telegram-бот не потребує інсталяції, завжди під рукою у користувача й може миттєво надавати навчальний контент.

Для реалізації Telegram-бота у даній дипломній роботі обрано стек технологій JavaScript та Node.js, а також бібліотеку Telegraf для взаємодії з Telegram API. У якості системи зберігання даних використано MongoDB. Особливу увагу приділено реалізації інтервального повторення, яке є одним із найефективніших способів запам’ятовування іноземних слів.

Мета дипломної роботи полягає у розробці Telegram-бота, що дозволяє користувачам ефективно вивчати іноземну лексику шляхом додавання власних слів, організації повторень, ведення статистики та практики в інтерактивному форматі. У процесі реалізації розглянуто переваги та обмеження чат-ботів у порівнянні з мобільними застосунками, виконано аналіз існуючих рішень, описано використані технології та наведено результати розробки.

1 АНАЛІЗ ТЕХНОЛОГІЙ ДЛЯ ВИВЧЕННЯ ІНОЗЕМНИХ СЛІВ

1.1 Порівняння форматів: мобільний застосунок чи чат-бот

Існує кілька рішень для створення систем вивчення іноземних слів, серед яких найбільш популярними є мобільні додатки та чат-боти. Кожен із цих підходів має свої переваги, особливості реалізації та цільову аудиторію.

Мобільні додатки (рис. 1.1) довгий час були основним інструментом для навчання, особливо після появи онлайн-магазинів, таких як App Store та Google Play, які значно спростили процес завантаження та оновлення програмного забезпечення.

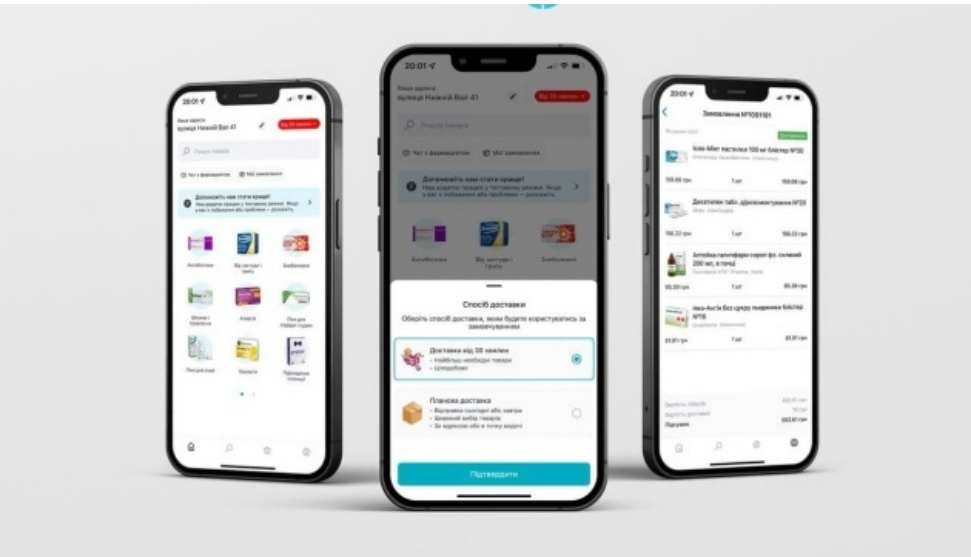


Рисунок 1.1 – Приклад мобільного додатку

Ці додатки зазвичай забезпечують зручний інтерфейс користувача, що дозволяє ефективно працювати з великими обсягами інформації. Крім того, вони часто підтримують офлайн-доступ, що є особливо корисним у випадках обмеженого або нестабільного інтернет-з'єднання. Функціонал мобільних додатків може включати інтерактивні вправи, аудіо- та відеоматеріали, систему повторення

з інтервалами (spaced repetition), а також гейміфікацію, що підвищує мотивацію користувача.

Однак у 2016 році стрімко зросла популярність чат-ботів (рис. 1.2), які стали ефективними помічниками у різних сферах, включаючи освіту. Це зумовлено широким розповсюдженням месенджерів, таких як Telegram, Facebook Messenger та Viber, через які користувачі вже звикли отримувати інформацію та спілкуватися. Завдяки інтеграції з такими платформами, чат-боти надають можливість миттєвого доступу до навчальних матеріалів без необхідності встановлення окремого додатку. Крім того, боти легко підтримують мультимедійний контент, швидко реагують на запити користувача та можуть бути доступними 24/7.

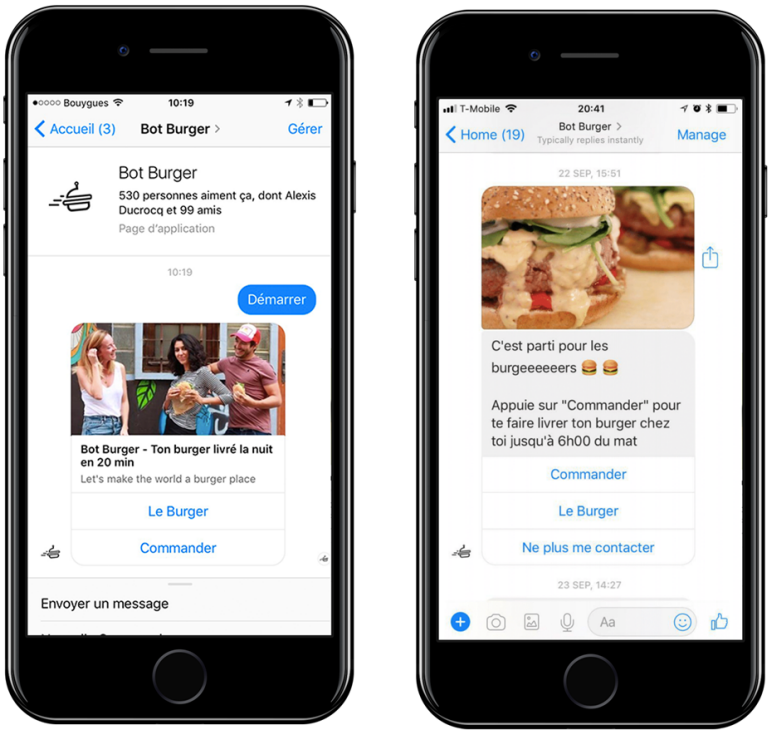


Рисунок 1.2 – Приклад чат-боту

Сучасні чат-боти стають дедалі розумнішими завдяки впровадженню алгоритмів машинного навчання. Це дозволяє їм аналізувати прогрес користувача, підлаштовувати темп і складність завдань, а також пропонувати індивідуальні плани навчання. Персоналізація навчального процесу сприяє підвищенню ефективності засвоєння нових слів та утриманню користувача в системі.

Вибір між мобільним додатком і чат-ботом залежить від конкретних вимог до продукту, технічних можливостей, бюджету на розробку та уподобань цільової аудиторії. Мобільні додатки більше підходять для комплексного навчання з великим функціоналом і гарною візуалізацією, тоді як чат-боти — для швидкого доступу, мінімалістичного інтерфейсу та навчання «на ходу». Тому важливо проаналізувати переваги та недоліки кожного підходу, щоб обрати найкраще рішення для реалізації поставлених цілей (табл. 1.1).

Таблиця 1.1 – Порівняння мобільних додатків і чат-ботів

Критерій	Мобільні додатки	Чат-боти
Доступність	Потрібна інсталяція з магазину	Не потребує встановлення
Інтерфейс	Повноцінний, з елементами гейміфікації	Мінімалістичний, текстовий або кнопковий
Функціональність	Широка: тести, відео, вимова, форуми	Обмежена, але достатня для базового навчання
Офлайн-режим	Часто підтримується	Потребує постійного інтернет-з'єднання
Адаптивність / AI	Є, особливо в популярних додатках	Може реалізовуватись через ML або NLP
Інтеграція у повсякденність	Потрібно окремо запускати	Вбудований у месенджер, завжди доступний
Складність у розробці	Вища, потребує UI/UX дизайну	Нижча, особливо з бібліотеками на кшталт Telegraf
Можливість швидкого оновлення	Через оновлення в App Store / Google Play	Оновлення коду бота — миттєве

1.2 Поняття та особливості чат-ботів

Чат-бот – це технологія взаємодії з користувачем, яка допомагає в обслуговуванні клієнтів, взаємодії та підтримці, замінюючи або доповнюючи операторів підтримки штучним інтелектом (AI) та іншими технологіями автоматизації, які можуть спілкуватися з кінцевими користувачами через чат. У цій статті детально пояснюється значення чат-ботів, їхні функції та типи з прикладами.

Багато компаній зараз використовують чат-боти для автоматизації взаємодії з користувачем і функцій транзакцій. Організації зазнали значної економії коштів і стали більш ефективними, оскільки зменшили свою залежність від допоміжного персоналу та живих операторів.

Чат-боти — це комп’ютерні програми, які відтворюють і аналізують людський діалог (усний чи письмовий), що дозволяє людям спілкуватися за допомогою електронних пристроїв так, ніби вони спілкуються з живою людиною. Чат-боти можуть варіюватися від простих програм, які реагують на запрограмовану логіку, до просунутих віртуальних помічників, які можуть навчатися та вдосконалюватись, збираючи та обробляючи дані, щоб забезпечити найвищий рівень персоналізації.

Чат-боти бездоганно інтегровані в кілька наших щоденних робочих процесів. Наприклад, ви можете переглядати платформу електронної комерції, щоб придбати товар на своєму комп’ютері, коли на моніторі з’являється вікно із запитом, чи потрібна вам допомога. Крім того, людина може використовувати голосовий ввід, щоб замовити напій у найближчій торговій точці та отримати сповіщення про те, коли замовлення буде готове та скільки воно коштуватиме.

Поширення чат-ботів за останні роки пов’язане з прискореними темпами цифрової трансформації. Підприємства все частіше переходять від традиційних способів зв’язку до цифрових каналів, щоб взаємодіяти та здійснювати операції зі своїми клієнтами. Підприємства використовують штучний інтелект (ШІ), щоб розблокувати нові можливості для ефективнішої взаємодії з клієнтами, а чат-боти є одними з найпопулярніших застосувань ШІ на підприємстві.

Згідно з прогнозами Gartner на 2022 рік, до кінця цього року 70% офісних співробітників щоденно взаємодіятимуть із чат-ботами та платформами спілкування. Це варіюється від роботів-консультантів для особистого користування (наприклад, Google Assistant і Alexa від Amazon) і чат-ботів, інтегрованих у програми обміну повідомленнями, такі як Telegram, Viber, Facebook Messenger і WeChat.

Бізнес може отримати вигоду від чат-ботів, оскільки вони підвищують продуктивність і заощаджують витрати, забезпечуючи зручність для клієнтів і пропонуючи додаткові послуги внутрішньому персоналу, клієнтам і партнерам. Вони дозволяють компаніям швидко відповідати на різноманітні питання серед зацікавлених сторін, одночасно зменшуючи потребу в людській участі.

Компанії можуть масштабувати, персоналізувати досвід і бути проактивно доступними за допомогою чат-бота, ключової відмінності в цифрову еру. Наприклад, коли бізнес покладається виключно на людські зусилля, він може обслуговувати лише певну кількість людей одночасно – що обмежує можливості та обмежує зростання. Компанії з процесами, які потребують інтенсивних зусиль вручну, змушені покладатися на дуже жорсткі моделі, щоб бути економічно ефективними, що означає, що їхні проактивні та індивідуальні можливості охоплення обмежені.

З іншого боку, чат-боти дозволяють компаніям взаємодіяти з практично нескінченною кількістю клієнтів у персоналізований спосіб, який можна збільшувати або зменшувати відповідно до поточних вимог. Можна створити майже “людське” обслуговування, адаптоване до кожної людини, навіть якщо чат-бот розгортається для мільйонів клієнтів одночасно.

Щоб зрозуміти, як працює чат-бот, ми повинні спочатку розглянути три основні механізми, що керують технологією. Трьома механізмами, які потребують вашої уваги, є процеси, засновані на правилах, прийняття рішень на основі ШІ та втручання живих операторів. Залежно від механізму чат-бота його функціональність дещо відрізнятиметься.

Програмне забезпечення чат-бота на основі правил виконує попередньо запрограмовану поведінку на основі “логіки сценаріїв”, яку ви створюєте на конструкторі чат-ботів. Подібно до цифрового помічника, технологія чат-бота на основі правил може поводитися певним чином залежно від дій натискання та простих ініціаторів подій, таких як натискання “так” або “ні”. Він також може виявити конкретне ключове слово чи комбінацію фраз (але лише за наявності точної відповідності).

Наприклад, ви можете запрограмувати чат-бота на основі правил відповідати не лише тоді, коли хтось вибере “чорний” або “білий” але також якщо вони кажуть “Я хочу чорний товар” – серверний модуль чат-бота відповідатиме терміну "чорний" за попередньо налаштованим правилом.

Чат-боти зі штучним інтелектом використовують штучний інтелект і технологію обробки природної мови (NLP), щоб розпізнавати структуру речень, інтерпретувати знання та вдосконалювати свою здатність відповідати на запитання.

Замість того, щоб покладатися на заздалегідь запрограмовану відповідь, чат-боти ШІ спочатку визначають, що говорить клієнт або користувач. Потім, коли вони з’ясували, що шукає користувач, чат-бот надає відповідь, яку він вважає правильною на основі доступних даних. Машина вивчає “правильні” відповіді з часом, аналізуючи правильні та помилкові відповіді.

Завдяки механізмам прийняття рішень на основі штучного інтелекту чат-бот може бути надзвичайно ефективним за умови, що він добре розуміє вашу організацію, її клієнтів і контекст. Ця функція в основному використовується великими підприємствами, такими як електронна комерція, а також іншими великими підприємствами, які вимагають масштабування.

Як працюють чат-боти, які використовують взаємодію з живим оператором?

Чат у прямому ефірі – це тип системи чату, яка розміщена на веб-сторінці або у вашому мобільному додатку та працює як вікно споживача для вашої команди підтримки та контактного центру. Використовуючи цей механізм, чат-боти включають можливості маршрутизації для спілкування оператора з клієнтом у режимі реального часу.

Коли клієнту потрібно поспілкуватися з представником вашої команди, чат-бот визначає доступність оператора та відповідно направляє запит на підключення. Це з'єднає клієнта з кимось, хто може допомогти йому вирішити його проблему – тобто оператором з відповідними навичками та знаннями. Чат-бот також сповіщає оператора, коли є запит клієнта, і інформує клієнта про дані оператора, такі як його ім'я, час очікування тощо.

Як бачите, ці процеси відносно зрозумілі, враховуючи те, що прогрес у технології чат-ботів сьогодні нескінченний і легкодоступний для користувачів і розробників.

1.3 Класифікація чат-ботів за функціональністю

Сотні тисяч компаній у всьому світі розробляють різноманітні форми чат-ботів, щоб покращити обслуговування клієнтів. У цьому розділі пояснюється різні типи чат-ботів, для чого вони використовуються та яке програмне забезпечення чат-ботів може бути найбільш корисним для вашої компанії[6].

Голосовий бот — це канал зв'язку голосу в текст і тексту в мовлення, який підтримує ШІ та розуміння природної мови (NLU). Технологія штучного інтелекту допомагає ідентифікувати ключові мовні сигнали та визначати оптимальну реакцію розмови. Механізм перетворення тексту в мовлення (TTS) згодом завершує взаємодію, перетворюючи повідомлення на аудіо або голос.

Ці боти запрограмовані на завершення всього процесу розуміння мовлення та відповіді у людський спосіб. Голосові помічники або голосові чат-боти надають складну модель спілкування, яку можна швидко застосувати в різних інструментах обслуговування клієнтів, включаючи інтерактивну голосову відповідь (IVR), самообслуговування та онлайн бази знань.

Гібридний чат-бот — це гармонійне поєднання чат-бота та живого чату, яке поєднує найкраще з обох боків. Представник служби підтримки клієнтів буде доступний у чаті, щоб відповісти на будь-які запитання клієнтів, які можуть бути надто складними чи деталізованими для автоматизації.

Компонент штучного інтелекту в чат-боті копіює розмови на основі того, як він запрограмований і потреб розмови. З іншого боку, гібридний чат-бот ініціює автоматизовану бесіду в чаті та намагається вирішити запит користувача якомога швидше та простіше. Якщо він не працює належним чином, представник служби підтримки клієнтів може втрутитися в будь-який момент або в предметну область, де чат-бот не може виконати завдання.

З появою нових інтерфейсів соціальних мереж організації тепер можуть розгортати алгоритм штучного інтелекту на всіх платформах обміну повідомленнями, яким віддають перевагу клієнти. Це включає Facebook Messenger, Twitter, Instagram, а також програми обміну повідомленнями, такі як Telegram, Viber, WhatsApp і WeChat. Це забезпечує приємнішу роботу в Інтернеті для клієнтів і підвищує залучення компанії – і все без збільшення робочого навантаження контакт-центру.

Найпростіший тип чат-бота, який використовується, базується на навігації за допомогою меню. У більшості випадків ці чат-боти дотримуються фіксованого дерева сценаріїв, яке відображається споживача у вигляді клікабельних кнопок. Ці чат-боти (наприклад, автоматичні меню клавіатури на телефонах, які ми використовуємо регулярно) просять користувача обрати варіант відповіді і натиснути відповідну кнопку, щоб отримати відповідь.

Хоча ці чат-боти підходять для відповідей на поширені запитання, на які надходить більшість запитів на підтримку, вони можуть не працювати в складніших ситуаціях. Важливо також зазначити, що чат-боти на основі меню найповільніше надають справжню цінність споживачеві, але вони прості та доступні для початку.

Чат-бот з API – це інший вид бота, який може виконувати певний набір завдань, якщо ви розширите його можливості за допомогою API. Наприклад, чат-бот може надавати інформацію про погоду, вимикати світло у вашій кімнаті, коли він підключений до розумного домашнього приладу, замовляти продукти онлайн тощо. Маючи доступ до вихідного коду API, розробники можуть створювати власні чат-боти та інтегруйте їх з іншими платформами.

Чат-боти на основі ключових слів можуть слухати, що вводять відвідувачі, і правильно відповідають на них, на відміну від чат-ботів на основі меню. Ці чат-боти використовують настроювані ключові слова та NLP, щоб виявити тригери дій у розмові, щоб зрозуміти, як правильно реагувати на споживача. Однак, зіткнувшись із багатьма подібними запитами, ці чат-боти можуть не впоратися. Чат-боти можуть не працювати, якщо ключові слова повторюються в численних пов'язаних запитах.

Ось чому чат-боти, які поєднують визначення ключових слів і навігацію за допомогою меню чи кнопок, стають дедалі популярнішими. Якщо функція визначення ключового слова не працює або користувачеві потрібна додаткова допомога у пошуку відповіді, такі чат-боти дають користувачам можливість безпосередньо вводити команди за допомогою навігаційних кнопок. Це ефективний обхідний шлях, коли бот не може виявити ключові слова у введеному тексті.

Чат-бот на основі правил ідеально підходить для компаній, які вже знають типи запитів, які ставитимуть їхні клієнти. Потоки чату створюються за допомогою логіки if/then, і ви повинні спочатку встановити вимоги до мови чат-бота. Умови оцінювання слів, структура слів, синоніми тощо є основними положеннями його функціональності. Клієнти отримають швидку допомогу, якщо вхідний запит відповідає цим параметрам.

Важливо зазначити, що розробник несе відповідальність за охоплення кожної перестановки та комбінації запиту, наскільки це можливо – інакше чат-бот не зможе зрозуміти споживача чи відповісти йому.

Контекстні чат-боти можуть зрозуміти контекст чату та визначити правильне значення запиту користувача. Він також може згадати попередні взаємодії та використовувати цю інформацію для підтримки актуальності під час взаємодії з постійними клієнтами. Контекстні боти можуть гарантувати, що постійні користувачі матимуть стабільний досвід. Крім того, він може зберігати інформацію про наміри користувача, зібрану з численних платформ і каналів, забезпечуючи відповідність контексту розмови потребам споживача в кожній точці взаємодії.

Контекстні чат-боти підключаються до централізованої бази даних сайту чи програми, як правило, системи керування взаємовідносинами з клієнтами (CRM) або платформи даних про клієнтів (CDP). Це дає їм змогу отримувати важливу інформацію про особу, з якою вони спілкуються, наприклад її ім’я, місцезнаходження чи історію покупок.

Чат-боти підтримки — це системи спілкування, призначені виключно для надання підтримки і послуг клієнтам після покупки. На відміну від ботів у соціальних мережах або на веб-сайтах, вони не діляться пропозиціями, рекламними акціями чи іншими матеріалами для залучення клієнтів. Цей тип чат-бота зазвичай зустрічається на порталах самообслуговування та в онлайн-документації, куди користувачі можуть звернутися за підтримкою та допомогою. Чат-боти підтримки широко використовуються для внутрішніх цілей, зокрема для відповідей на запити відділу кадрів, збору запитів у IT, подання документів співробітникам тощо.

Транзакційні чат-боти можуть допомогти організаціям зібльшити продажі та маркетингові зусилля, чи то для планування зустрічей, створення потенційних клієнтів або збору платежів. Користувачі можуть здійснювати транзакції безпосередньо під час спілкування з чат-ботом без втручання людини.

Його найбільша перевага — це можливість здійснювати транзакції та розвивати бізнес у режимі 24/7/365. Як наслідок, транзакційний чат-бот відрізняється від інших типів ботів, таких як інформаційні чи боти підтримки. Його зосереджено на завершенні транзакції та оптимізації взаємодії з користувачем, пропонуючи швидкий і простий канал для однієї мети. Його розроблено для виконання невеликої кількості спеціалізованих завдань.

Чат-боти традиційно проектувалися та розроблялися з використанням коду для створення дерев рішень та використання алгоритмів штучного інтелекту й машинного навчання, що забезпечують технологію. Кожна мова програмування має веб-API, який можна використовувати для створення чат-ботів. Окрім PHP і Node.js, у більшості типових розгортань використовується багато інших бібліотек, які підтримують Python або Java.

Однак останні досягнення дозволяють організаціям використовувати чат-ботів, які потребують незначного програмування або взагалі не потребують його. Це забезпечує швидшу роботу додатків і генерацію цінності, оскільки для створення та налаштування бота доступний графічний інтерфейс користувача (GUI). Розгортання без коду підходять для чат-ботів, які збирають інформацію, і тих, які заохочують взаємодію людей. Навпаки, чат-боти з низьким кодом ідеально підходять для організацій, яким потрібно додати унікальні функції, одночасно зменшуючи зусилля щодо розробки.

1.4 Аналіз існуючих рішень для вивчення іноземних мов

RepeatWordBot (рис.1.3) – це бот для вивчення іноземних слів за допомогою методу інтервальних повторень (Spaced Repetition System, SRS). Він допомагає ефективно запам’ятовувати слова, нагадуючи про повторення у оптимальні моменти[12].

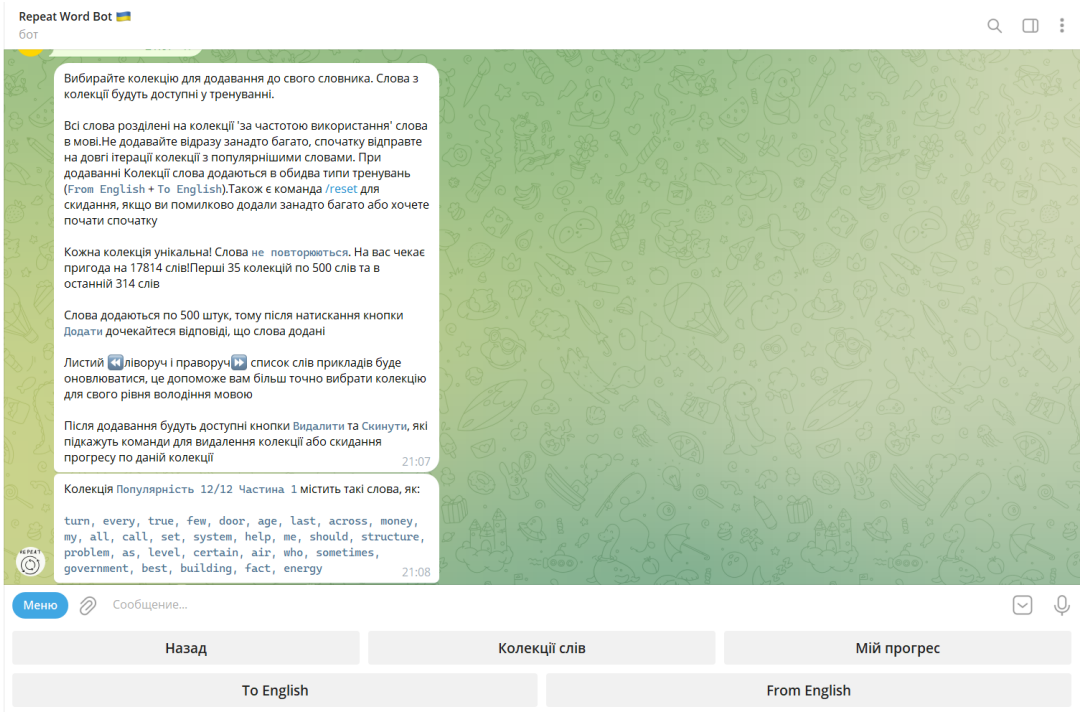


Рисунок 1.3 – Бот для вивчення іноземних слів RepeatWordBot

Основні функції бота:

— Готові списки слів – можна вибирати з доступних наборів (англійська, німецька, іспанська тощо).

— Додавання власних слів – можна створювати свої списки для вивчення.

— Інтервальне повторення – бот автоматично нагадує, коли треба повторити слова.

— Режим тренування – картки з перекладом, тести на запам’ятовування.

— Статистика – відстеження прогресу в навчанні.

Як це працює:

— Активація бота: запускаєте @RepeatWordBot у Telegram.

— Вибір списку слів: берете готовий (наприклад, "Топ-1000 англійських слів") або додаєте власний.

— Вивчення: бот показує слова з перекладом і прикладами.

— Повторення: система саме визначає, коли вам треба повторити слово (на основі вашого рівня запам’ятовування).

— Прогрес: бот надає статистику, скільки слів ви вивчили.

Переваги:

— Безкоштовний – немає платних підписок.

— Простий інтерфейс – зрозумілий навіть для новачків.

— Ефективна методика – інтервальні повторення допомагають запам’ятовувати слова надовго.

— Гнучкість – можна вивчати будь-яку мову, додаючи свої слова.

Недоліки:

— Дизайн може бути простим – без сучасного інтерфейсу, як у Quizlet чи Anki.

RepeatWordBot – хороший вибір для тих, хто хоче вивчати слова системно, без зайвих складнощів, шукає простий SRS-тренажер прямо в Telegram, не хоче встановлювати додаткові додатки.

Quizlet (рис.1.4) – це один із найпопулярніших сервісів для вивчення іноземних слів, термінів та іншої інформації за допомогою флеш-карток, ігор і тестів. Він використовує метод інтервальних повторень та інтерактивні вправи для ефективного запам’ятовування[13].

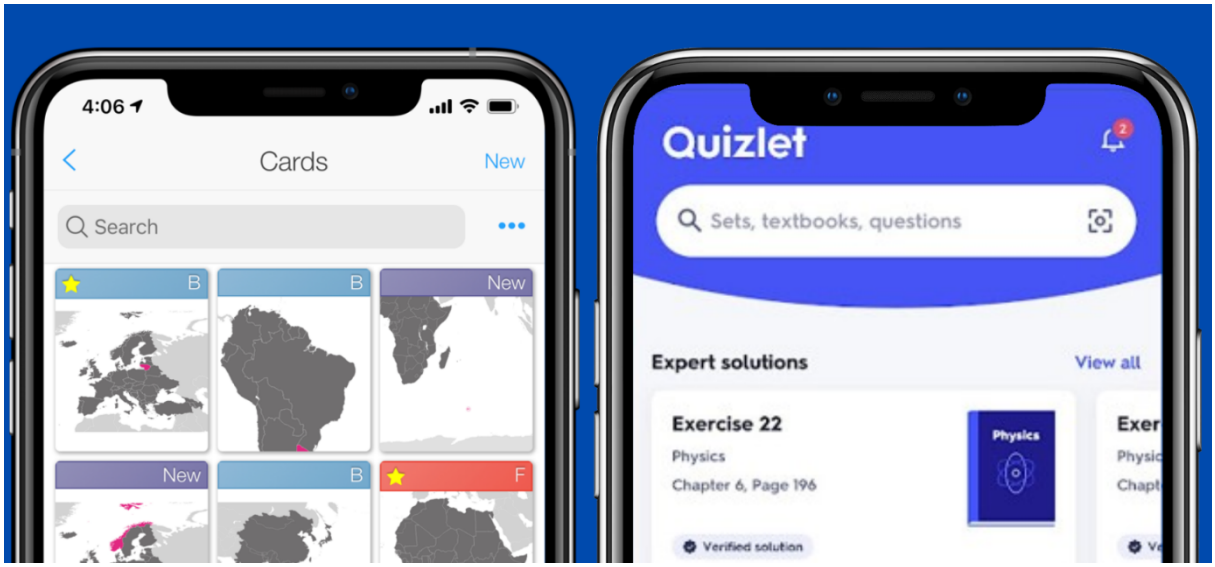


Рисунок 1.4 – Додаток для вивчення іноземних слів Quizlet

Основні функції Quizlet

— Створення та використання флеш-карток

— Власні набори – можна додавати слова з перекладом, визначеннями, зображеннями та аудіо.

— Готові набори – мільйони публічних наборів з різних мов і предметів.

— Різні режими навчання: картки (Flashcards) – класичне засвоєння слів, написання (Write) – введення слова правильно за пам’яттю, тестування (Test) – автоматично генерується тест із різними типами завдань, гра "Збіжки" (Match) – швидкий режим на знаходження пар слів, гра "Гравітація" (Gravity) – аркадна гра для тренування словникового запасу, інтервальне повторення (Quizlet Learn), автоматичний підбір слів для повторення на основі вашого прогресу, групи та класи (для навчання в команді), а також можна створювати класи, ділитися наборами слів і відстежувати прогрес учнів.

Переваги Quizlet:

— Простота – інтуїтивний інтерфейс.

— Багато готових наборів – не треба створювати все з нуля.

— Інтерактивні ігри – навчання через гру.

— Доступ на різних пристроях – веб, iOS, Android.

— Функція аудіо – можна слухати вимову слів.

Недоліки Quizlet:

— Без платної підписки обмежені функції (наприклад, деякі режими навчання).

— Деякі набори можуть бути неточними (оскільки їх створюють користувачі).

2 ВИБІР І ОПИС ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ БОТА

2.1 Середовище виконання JavaScript: Node.js

Node.js (рис.2.1) — це однопоточне кросплатформенне середовище виконання з відкритим вихідним кодом і бібліотека, яка використовується для запуску вебзастосунків, написаних на JavaScript, поза браузером клієнта.

Рисунок 2.1 – Середовище виконання JavaScript: Node.js

Простіше кажучи, Node.js — це програмне середовище, яке дозволяє запускати програми, написані мовою JavaScript, поза межами браузера. Історично склалося так, що програми, написані на JavaScript, на відміну від інших мов програмування, можна було запускати лише в браузерах, які мали вбудований рушій виконання цього коду. Поза браузером JavaScript, можна сказати, не працював[1].

Під час розробки Node.js за основу був узятий рушій виконання JavaScript під назвою V8 (рис.2.2), створений компанією Google та використаний у браузері Google Chrome. Завдяки Node.js код JavaScript тепер можна запускати фактично в будь-якому середовищі, тому з використанням цієї бібліотеки можна писати не лише фронтенд, а й серверну частину вебзастосунку.

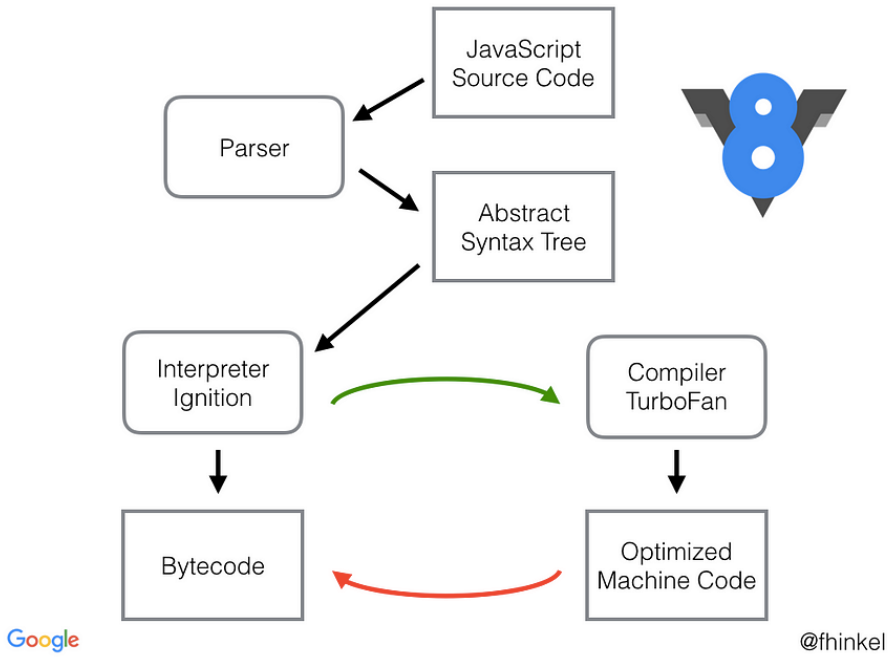


Рисунок 2.2 – Рушій виконання JavaScript під назвою V8

Простіше кажучи, це означає, що цілі сайти тепер можуть працювати на єдиному «стеку» технологій, що значно пришвидшує розробку і полегшує підтримку, дозволяючи зосередитися на досягненні бізнес-цілей проєкту.

Node.js має відкритий вихідний код, тому працювати з ним можна абсолютно безкоштовно. Його й досі активно розвиває та вдосконалює глобальна спільнота розробників.

Варто розуміти, що Node.js насправді не є фреймворком і не бібліотекою, як це буває у випадку з традиційним програмним забезпеченням — це середовище виконання. Воно легке, гнучке і просте в розгортанні, а всі його функції спрямовані на оптимізацію та прискорення роботи застосунку[8].

Node.js (рис.2.3), був створений у 2009 році Райаном Далем. Даль критикував обмеження, притаманні популярним на той час вебсерверам. Раніше сервери із труднощами обробляли велику кількість одночасних з'єднань, а виконання програм або блокувало весь процес, або вимагало багатоядерного процесора. Усі ці проблеми заважали компаніям створювати універсальні продукти для великої кількості користувачів.

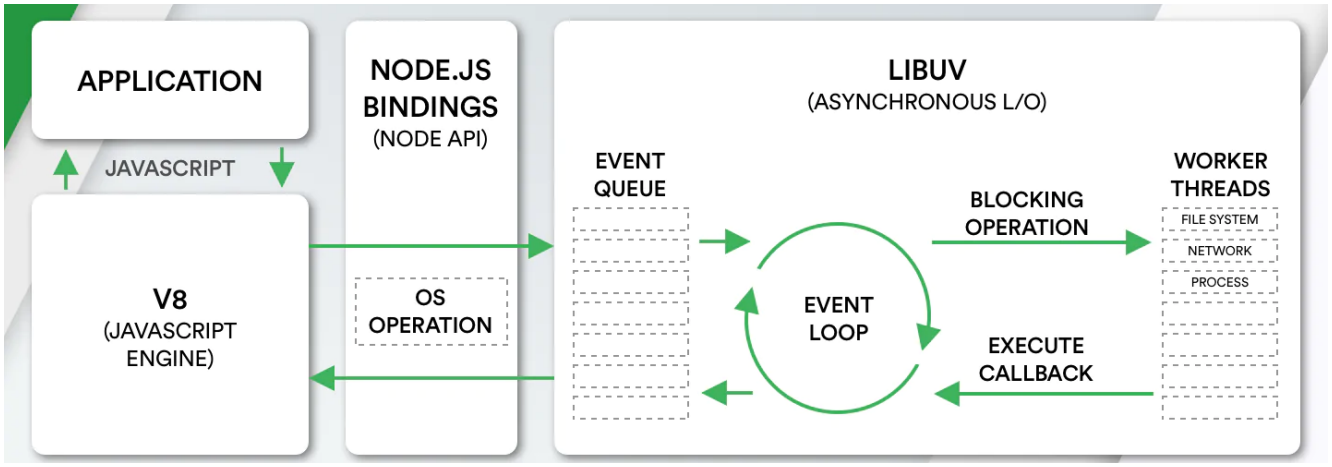


Рисунок 2.3 – Архітектура NodeJs

У відповідь на це Даль створив Node.js, щоб надати розробникам можливість використовувати JavaScript для створення серверної частини застосунків, фактично уніфікувавши розробку вебзастосунків навколо єдиної мови програмування.

Перша версія Node.js підтримувала лише Linux та Mac OS X. Спочатку її розробкою керував сам Даль, а згодом підтримку взяла на себе компанія Joyent, що спеціалізується на програмному забезпеченні та супровідних послугах[10].

У січні 2010 року був показаний менеджер пакетів (рис.2.4) для Node.js (npm), який значно спростив публікацію та спільне використання вихідного коду, а також встановлення, видалення й оновлення програм.

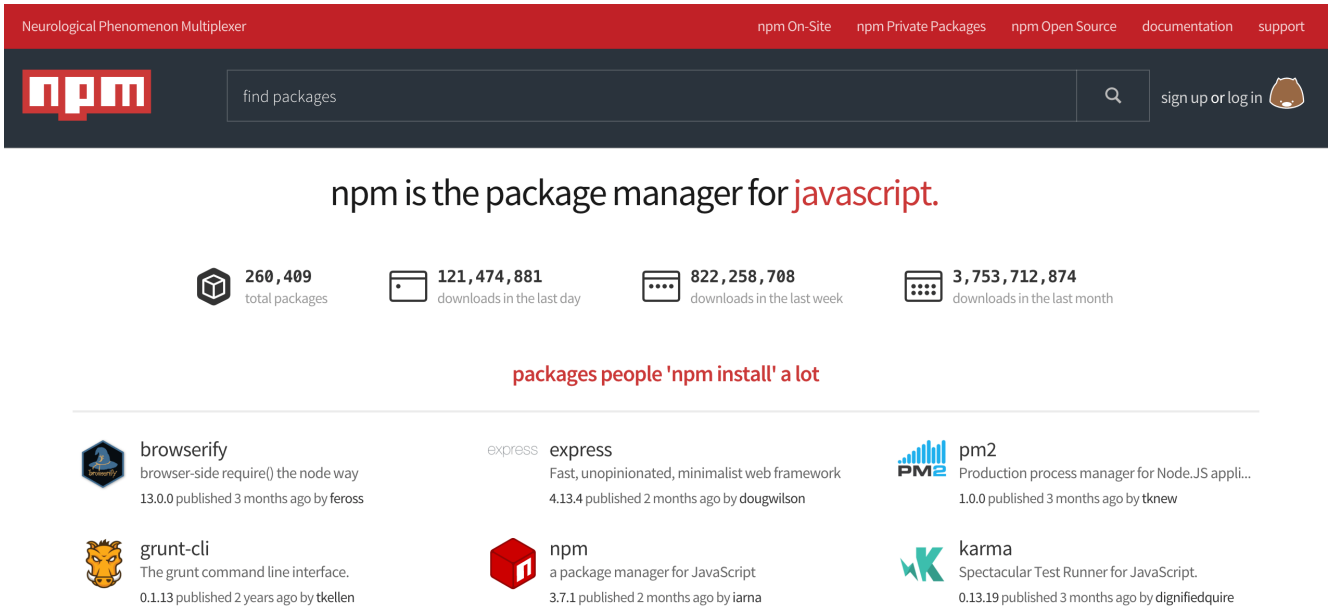


Рисунок 2.4 – Менеджер пакетів для Node.js (npm)

У 2011 році Microsoft і Joyent об’єдналися для створення версії Node.js для Windows, що значно розширило список підтримуваних операційних систем і надало розробникам більше можливостей.

Згодом було створено Node.js Foundation, що об’єднало спільноту розробників. У 2019 році цей фонд об'єднався з JS Foundation, утворивши нову організацію — OpenJS Foundation.

Node.js — одна з найкращих технологій, якими може користуватись розробник. Крім простоти у використанні, вона також дуже швидка, забезпечуючи миттєвий відгук на запити клієнтів.

Основні характеристики:

— Реалізований на JavaScript Node.js написаний мовою JavaScript — найпопулярнішою мовою програмування у світі. Більшість розробників вже знайомі з JavaScript, його принципами та концепціями. Це робить Node.js простим для вивчення та використання. Оскільки JavaScript також використовується у фронтенді, розробники можуть створювати повноцінні вебзастосунки, знаючи лише одну мову.

— Важливою особливістю Node.js є асинхронний характер. Це означає, що сервер не чекає завершення кожного запиту — декілька процесів можуть виконуватись паралельно. Крім того, він має неблокуючий ввід/вивід, що значно покращує швидкість роботи.[9]

— «Керування подіями» означає, що виконання коду відбувається після настання певної події. У Node.js функції зворотного виклику (callback-функції) відіграють важливу роль. Вони не лише спрощують обробку, а й споживають менше ресурсів.

— Однопоточна модель Node.js працює в одному потоці та використовує циклічну обробку подій (Event loop). Завдяки цьому значно скорочується час на обробку запитів і зменшується навантаження на сервер.

— Кросплатформеність Node.js працює на багатьох операційних системах від Windows і Mac до Linux і мобільних платформ. Це дозволяє створювати застосунки в будь-якому середовищі[15].

— Швидка потокова передача даних Node.js використовує високошвидкісний JavaScript-рушій V8, який також використовується в Google Chrome. Це забезпечує швидке виконання коду та миттєву обробку потоків даних у вебзастосунках.

2.2 Система керування базами даних MongoDB та бібліотека Mongoose

MongoDB це документно-орієнтована база даних NoSQL (рис.2.5), яка використовується для зберігання великих обсягів даних. Замість використання таблиць і рядків, як у традиційних реляційних базах даних, MongoDB використовує колекції та документи. Документи складаються з пар ключ-значення, які є основною одиницею даних у MongoDB. Колекції містять набори документів і функції, що є еквівалентом таблиць реляційної бази даних. MongoDB це база даних, яка з'явилася на світ приблизно в середині 2000-х років.

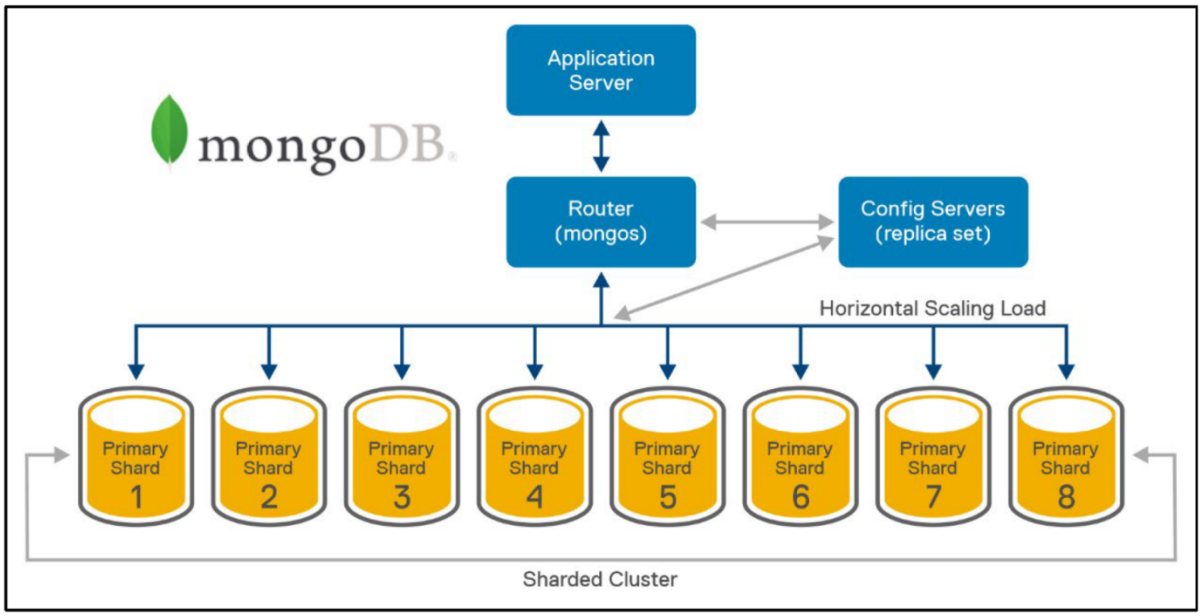


Рисунок 2.5 – Архітектура MongoDB

Кожна база даних містить колекції, які, у свою чергу, містять документи. Кожен документ може мати різну кількість полів. Розмір і зміст кожного документа можуть відрізнятися один від одного[7].

Структура документа більше відповідає тому, як розробники будують свої класи та об’єкти на відповідних мовах програмування. Розробники часто кажуть, що їхні класи не є рядками та стовпцями, а мають чітку структуру з парами ключ-значення.

Рядки (або документи, як викликано в MongoDB) не потребує попередньої визначеної схеми. Натомість поля можна створювати на льоту.

Доступна модель даних MongoDB дозволяє легше представляти ієрархічні зв’язки, зберігати масиви та інші складніші структури[2].

Масштабованість – The MongoDB середовища дуже масштабовані. Компанії в усьому світі визначили кластери, деякі з них працюють із 100+ вузлами з приблизно мільйонами документів у базі даних.

MongoDB приклад (рис.2.6) наведений нижче приклад показує, як можна моделювати документ MongoDB.

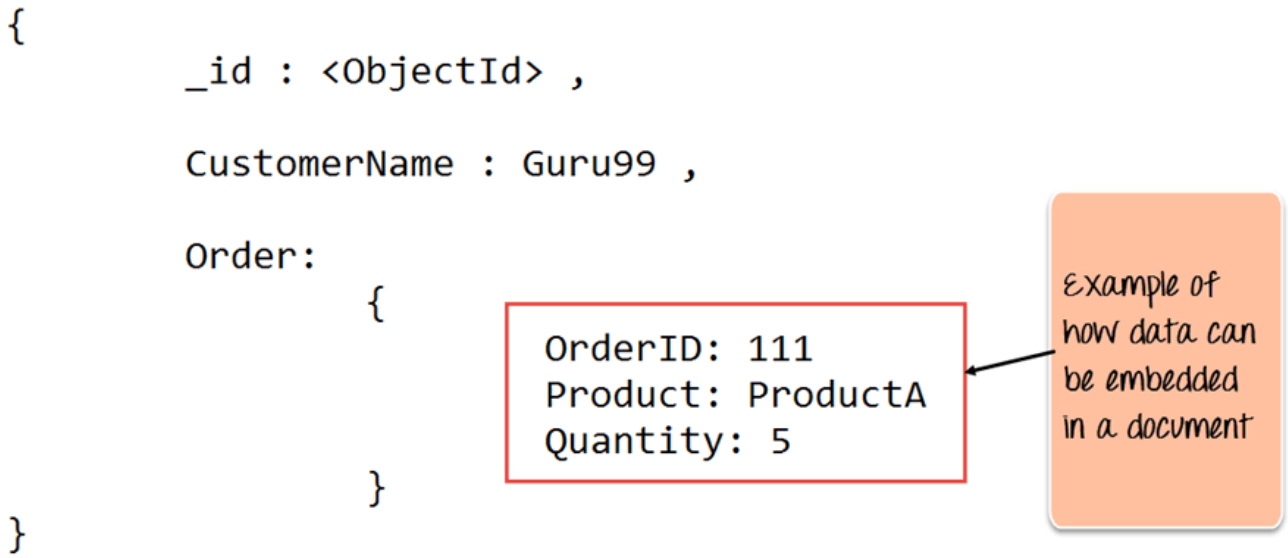


Рисунок 2.6 – Приклад MongoDB

Поле _id додається MongoDB для однозначної ідентифікації документа в колекції.

Однією з ключових відмінностей у моделюванні даних MongoDB є те, що дані замовлення (OrderID, Product і Quantity), які в RDBMS зазвичай зберігаються в окремій таблиці, в MongoDB фактично зберігаються як вбудований документ у

самій колекції.Нижче наведено кілька поширених термінів, які використовуються в MongoDB.

_id – це поле обов'язкове для кожного MongoDB документ. Поле _id представляє унікальне значення в MongoDB документ. Поле _id схоже на первинний ключ документа. Якщо ви створюєте новий документ без поля _id, MongoDB автоматично створить поле та додасть 24-значний унікальний ідентифікатор до кожного документа в колекції.

COLLECTION – це угруповання MongoDB документів. Колекція є еквівалентом таблиці, яка створюється в будь-якому іншому RDMS, наприклад Oracle або MS SQL. Колекція існує в одній базі даних. Як видно зі вступу, колекції не забезпечують дотримання будь-якої структури.

Курсор – це вказівник на набір результатів запиту. Клієнти можуть ітерувати курсор, щоб отримати результати.

Database – це контейнер для колекцій, як у RDMS, де це контейнер для таблиць. Кожна база даних отримує власний набір файлів у файловій системі. А MongoDB сервер може зберігати кілька баз даних.

Документ – запис в MongoDB колекцію в основному називають документом. Документ, у свою чергу, складатиметься з імені поля та значень.

Поле – пара ім'я-значення в документі. Документ має нуль або більше полів. Поля аналогічні стовпцям у реляційних базах даних. На наступній діаграмі показано приклад полів із парами «Ключ-значення». Отже, у наведеному нижче прикладі CustomerID і 11 є однією з пар ключів-значень, визначених у документі.

JSON (рис.2.7) – це відомо як JavaScript Нотація об'єктів. Це зрозумілий для людини формат звичайного тексту для вираження структурованих даних. Наразі JSON підтримується багатьма мовами програмування[11].

```
{
  "DocumentType": 1,
  "No.": "S-ORD101001",
  "SellToCustNo": "10000",
  "PostingDate": "2023-04-02",
  "Lines": [
    {
      "LineNo": 10000,
      "Type": 2,
      "No": "1996-S",
      "Quantity": 12,
      "UnitPrice": 1397.3
    },
    {
      "LineNo": 20000,
      "Type": 2,
      "No": "1900-S",
      "Quantity": 4,
      "UnitPrice": 192.8
    }
  ]
}
```


Рисунок 2.7 – Приклад JSON

Нижче наведено кілька причин, чому варто почати використовувати MongoDB

— Документоорієнтований – З MongoDB це NoSQL замість того, щоб мати дані у форматі реляційного типу, вона зберігає дані в документах. Це робить MongoDB дуже гнучкий і адаптований до ситуації та вимог реального ділового світу.

— Спеціальні запити – MongoDB підтримує пошук за полями, запитами діапазону та пошук за регулярними виразами. Можна створювати запити для повернення певних полів у документах.

— Індексування – індекси можна створювати для покращення ефективності пошуку всередині MongoDB. Будь-яке поле в a MongoDB документ можна проіндексувати[3].

— Реплікація – MongoDB може забезпечити високу доступність за допомогою наборів копій. Набір реплік складається з двох або більше екземплярів Mongo DB. Кожен член набору реплік може виступати в ролі основної або вторинної репліки в будь-який час. Основна репліка — це основний сервер, який взаємодіє з клієнтом і виконує всі операції читання/запису. Вторинні репліки

зберігають копію даних основної за допомогою вбудованої реплікації. Коли первинна репліка виходить з ладу, набір реплік автоматично перемикається на вторинний, а потім він стає основним сервером.

— Балансування навантаження – MongoDB використовує концепцію шардингу для горизонтального масштабування шляхом поділу даних на кілька MongoDB екземпляри. MongoDB може працювати на кількох серверах, балансуючи навантаження та/або дублюючи дані, щоб підтримувати працездатність системи у випадку апаратного збою.

2.3 Контейнеризація застосунку за допомогою Docker

Docker — це програмне забезпечення, призначене для створення ізольованих середовищ виконання (контейнерів), у межах яких можна інсталювати необхідну операційну систему, конкретну версію Java, налаштувати змінні середовища, а також встановити відповідні залежності. Доступ до такого середовища може бути наданий лише за заданих умов. Завдяки механізмам ізоляції, програмне забезпечення, що виконується в контейнері, функціонує незалежно від зовнішнього середовища, що забезпечує стабільність та передбачуваність його роботи.

Сервер — це обчислювальний пристрій, який зазвичай має вищу продуктивність порівняно зі звичайними персональними комп'ютерами. Він може бути фізичним або віртуальним і функціонує під керуванням певної операційної системи, наприклад Windows. У випадку, коли для запуску конкретного програмного забезпечення потрібне середовище Linux або специфічні версії операційної системи чи бібліотек, виникає проблема сумісності. Очевидно, що перевстановлення системи або придбання нового обладнання не є ефективним рішенням.

Контейнеризація — це технологічний підхід, який дозволяє ізолювати програмне забезпечення разом із усіма його залежностями в окреме середовище, так званий контейнер. Аналогічно до того, як вантажі транспортуються у стандартизованих контейнерах на вантажних суднах, програмне забезпечення

упаковується в програмні контейнери, що гарантує його цілісність, безпечність та незалежність від середовища виконання. У такій парадигмі сервер виконує роль транспортного засобу, тоді як Docker (рис. 2.8), виступає інструментом для створення, управління та запуску контейнерів.

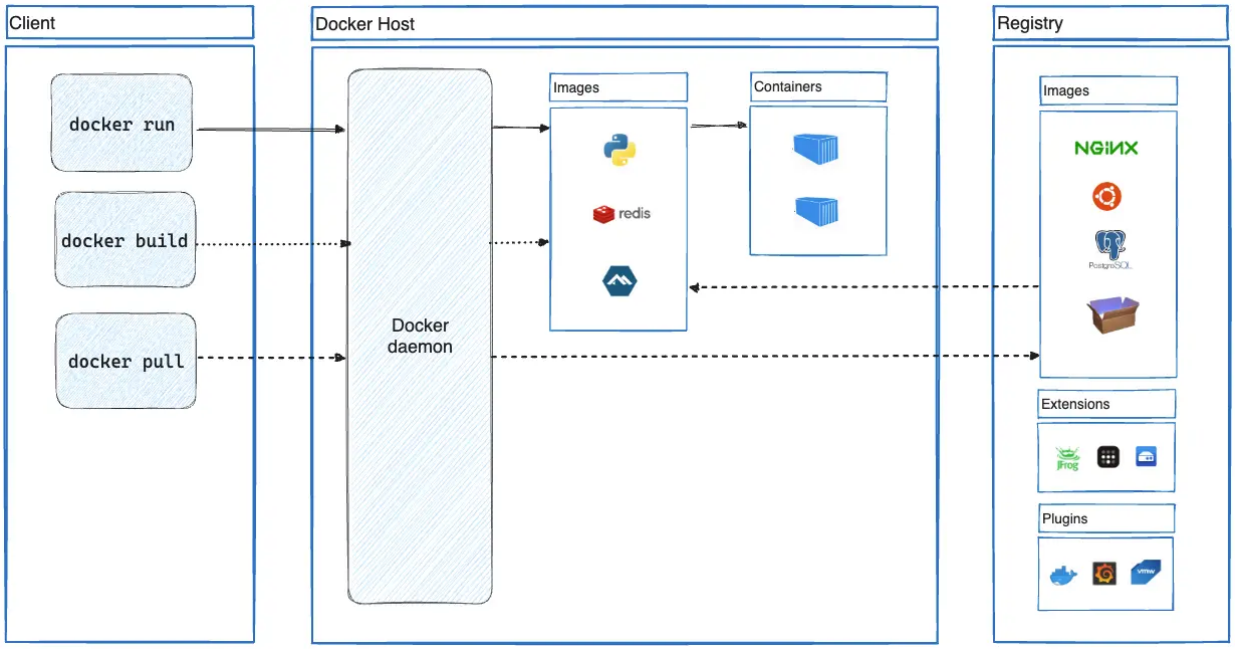


Рисунок 2.8 – Архітектура Docker

Завдяки цій ізоляції забезпечується взаємна незалежність програмних середовищ: у межах одного сервера може одночасно функціонувати кілька контейнерів з різними операційними системами — Windows, Linux або навіть MacOS. Цей підхід є відповіддю на зростаючу потребу запуску багатьох програм з різними конфігураціями в межах одного фізичного ресурсу.

Віртуальна машина (ВМ) — це програмна емуляція апаратного забезпечення, яка дозволяє запускати кілька операційних систем на одному фізичному пристрої. Утім, на відміну від традиційних ВМ, контейнери, що створюються за допомогою Docker, мають значно менший розмір, швидше запускаються та є більш гнучкими у налаштуванні. Наприклад, базовий образ Ubuntu може займати лише близько 68 МБ.

Однією з ключових переваг Docker є кросплатформність: контейнеризовані програми можна розробляти на одній операційній системі (наприклад, Windows), а

запускати на іншій (наприклад, MacOS або Linux) без необхідності змін у коді чи налаштуваннях. Схема створення контейнера (рис.2.9) виглядає наступним чином:

— Створюєте 'Dockerfile' — файл, в якому необхідно описати, як буде створюватися образ.

— Image — це образ, на підставі якого в подальшому буде запущений контейнер.

— Container — це запущений образ, в якому працює

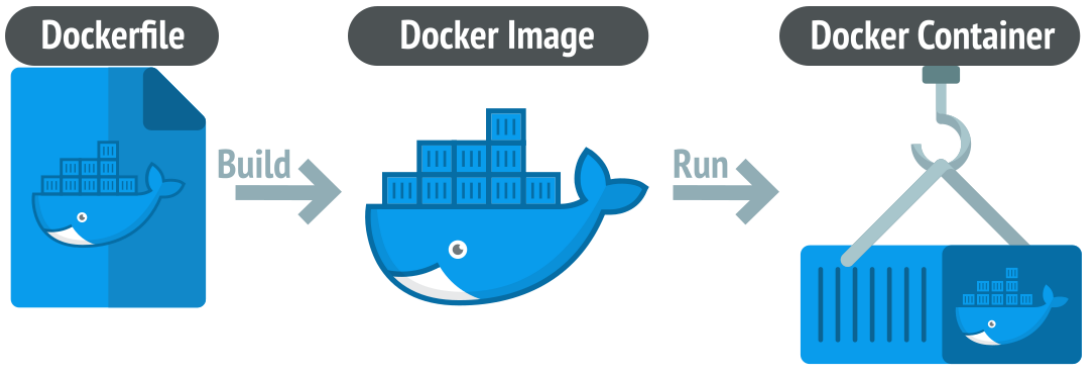


Рисунок 2.9 – Схема створення контейнера Docker

Варто розуміти, що програмне забезпечення та його залежності не з’являються у контейнері довільно. Їхнє автоматичне завантаження та встановлення забезпечується через взаємодію з віддаленими репозиторіями, в яких зберігаються необхідні компоненти. Docker, дотримуючись інструкцій користувача, автоматично виконує всі дії, пов’язані з пошуком, завантаженням і налаштуванням середовища виконання. Таким чином, відпадає потреба в ручному пошуку програм на різних сайтах — весь процес автоматизовано.

Docker надає централізований доступ до всіх необхідних компонентів для запуску програм.

Основні поняття, пов’язані з контейнеризацією в Docker:

Dockerfile — текстовий файл-інструкція, що описує алгоритм створення образу контейнера. Він не має обов’язкового розширення, проте його структура строго визначена синтаксисом Docker. У цьому файлі вказуються базовий образ,

необхідні залежності, налаштування середовища, команди для встановлення програмного забезпечення тощо.

Image (образ) — заздалегідь зібране середовище, створене на основі Dockerfile. Образи можуть бути завантажені з публічних або приватних репозиторіїв (наприклад, Docker Hub), що дає змогу повторно використовувати готові конфігурації без необхідності їх самостійного створення. Один і той самий образ може слугувати основою для запуску кількох контейнерів.

Container (контейнер) — запущений екземпляр образу, що є ізольованим середовищем виконання. Контейнери можна створювати, запускати, зупиняти або видаляти. У межах однієї системи можуть функціонувати як незалежні контейнери, так і ті, що взаємодіють між собою в рамках складнішої архітектури. Docker Engine (рис.2.10) — це ядро Docker, власне, це сам докер.

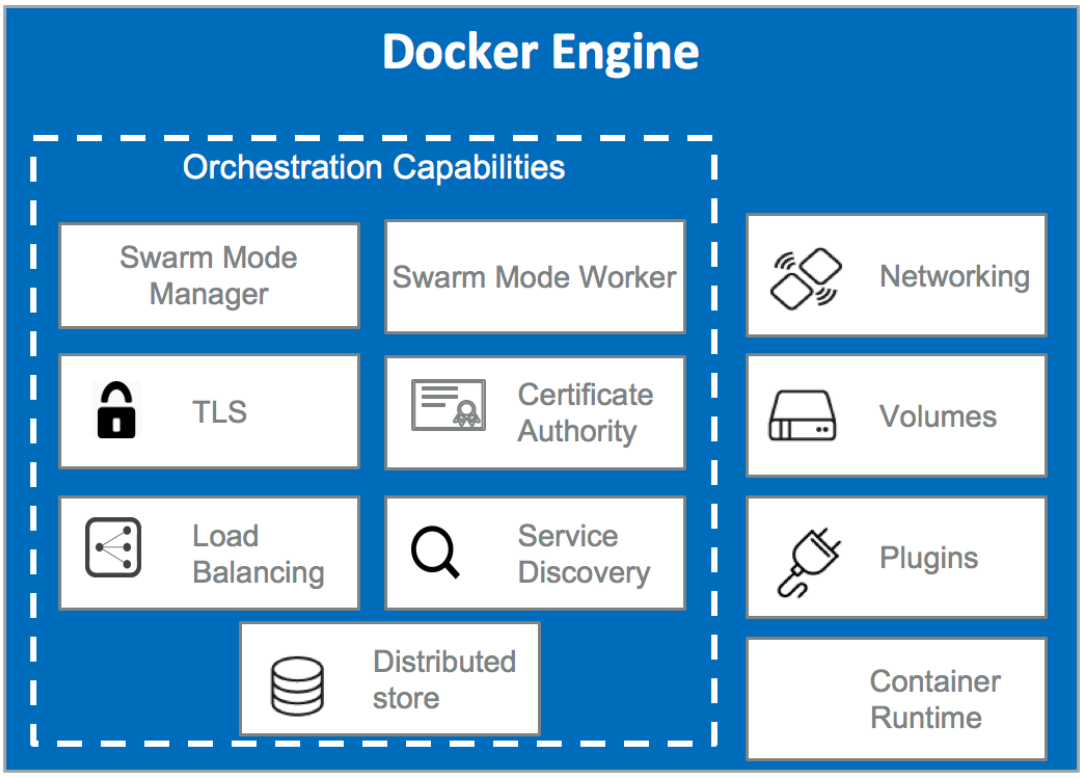


Рисунок 2.10 – Ядро Docker

Docker Hub — це публічний реєстр Docker-образів, який виконує функцію віддаленого сховища. Він дозволяє завантажувати готові образи операційних систем, середовищ виконання та програмного забезпечення, а також зберігати

власні зібрані образи з можливістю подальшого доступу до них з будь-якої точки світу. Це полегшує спільне використання контейнеризованих рішень та забезпечує повторюваність розгортання програмного середовища.

Docker Compose — це інструмент для опису і керування багатоконтейнерними додатками. З його допомогою можна визначити конфігурацію декількох взаємопов'язаних контейнерів у єдиному YAML-файлі (зазвичай docker-compose.yml). Docker Compose дозволяє використовувати налаштування, описані в Dockerfile, і забезпечує централізоване управління запуском, зупинкою та масштабуванням сервісів. Цей інструмент є особливо корисним у складніших проєктах, де необхідна координація кількох компонентів (наприклад, база даних, бекенд та фронтенд).

Загалом, технологія контейнеризації надає можливість створювати ізольовані середовища з індивідуальними конфігураціями, не впливаючи на глобальні налаштування операційної системи. Це дозволяє уникати конфліктів між програмами та зберігати стабільну роботу сервера навіть при розгортанні нових програмних модулів.

Популярність Docker зумовлена його ефективністю та гнучкістю, завдяки чому він активно використовується в комерційних і дослідницьких IT-проєктах. Вивчення базових принципів роботи з Docker є актуальним і рекомендованим навіть на ранніх етапах розробки, оскільки дозволяє формувати навички розгортання програм у професійному середовищі.

Docker Daemon (служба Docker) — фоновий процес, який функціонує на хост-системі та забезпечує управління контейнерами, образами, мережами і сховищами. Він надає REST API, яке використовується для взаємодії з іншими інструментами Docker, зокрема з клієнтською частиною.

Docker Client — інтерфейс користувача для взаємодії з Docker Daemon. Найпоширенішим клієнтом є CLI (інтерфейс командного рядка), проте також існують графічні інтерфейси, такі як Docker Desktop. Клієнт надсилає команди службі, яка виконує відповідні операції.

Docker Registry — централізоване сховище Docker-образів. Найпопулярнішим публічним реєстром є Docker Hub, однак існує можливість створення і використання приватних реєстрів для зберігання конфіденційних або внутрішньокорпоративних образів. До базових понять платформи належать:

— Контейнер — це ізольоване середовище виконання, яке містить програму та всі необхідні для її роботи залежності. Контейнер функціонує як незалежний процес у межах операційної системи хосту, але ізольований від інших процесів. Він забезпечує однакове виконання програмного забезпечення незалежно від середовища, у якому розгорнуто контейнер. Контейнери створюються на основі попередньо зібраних образів, що містять усі потрібні файли та конфігурації.

— Образ (Image) — це незмінний шаблон, який використовується для створення контейнерів. Він включає прикладний код, системні бібліотеки, середовище виконання, конфігураційні файли та всі інші компоненти, необхідні для запуску додатка. Образи є багаторазово використовуваними та можуть бути поширеними через реєстри, зокрема Docker Hub.

— Реєстр (Registry) — централізоване сховище, призначене для зберігання, керування та поширення Docker-образів. Реєстр дозволяє розробникам зберігати власні зібрані образи, отримувати доступ до публічних образів, а також ділитися ними з іншими користувачами або командами. Найвідомішим публічним реєстром є Docker Hub, однак також підтримується розгортання приватних реєстрів для внутрішнього використання.

— Dockerfile — це текстовий файл(рис.2.11), у якому послідовно описуються інструкції для створення Docker-образу. У цьому файлі зазначаються базовий образ, додаткові залежності, необхідні команди для встановлення програмного забезпечення, а також копіювання вихідного коду додатка до контейнера. На підставі Dockerfile виконується автоматизоване збирання образу, що забезпечує відтворюваність середовища розробки та розгортання.

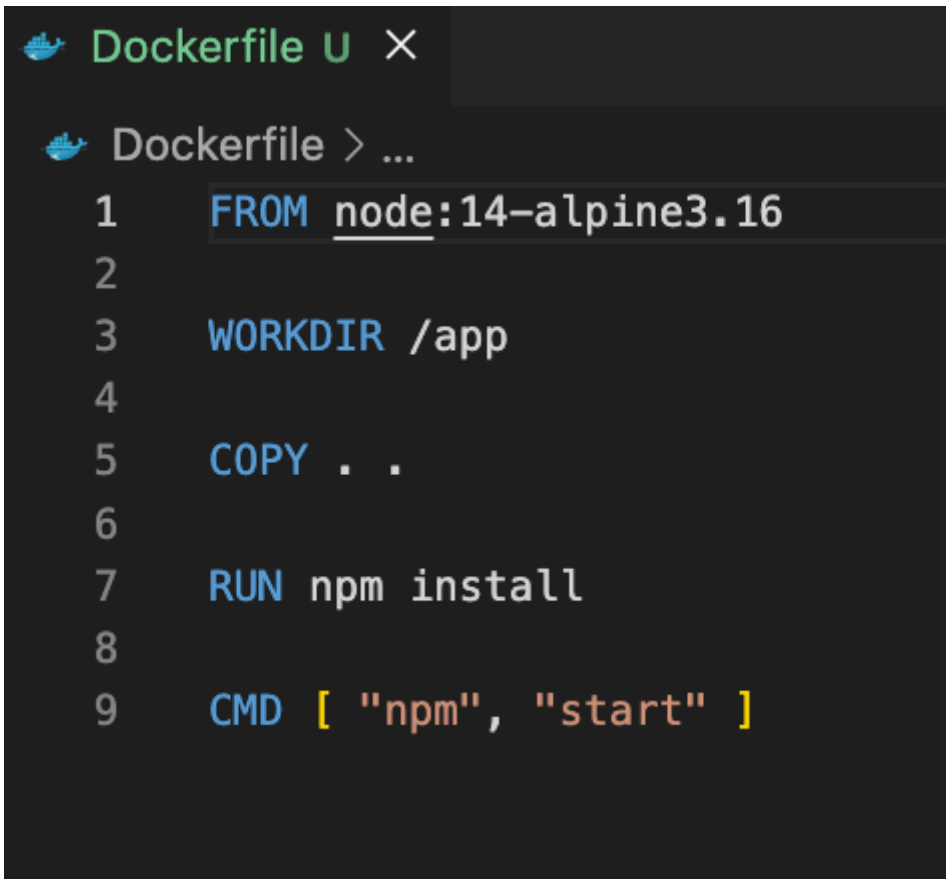


Рисунок 2.11 – Приклад текстового файлу Dockerfile

— Docker Compose — це інструмент для автоматизованого управління багатоконтейнерними застосунками. За допомогою конфігураційного файлу docker-compose.yml можна описати архітектуру застосунку: образи, залежності, змінні оточення, мережі та томи. Compose дозволяє запускати, зупиняти та масштабувати всі пов’язані сервіси як єдину систему однією командою, що значно спрощує процес розгортання.

— CLI (Command-Line Interface) — це інтерфейс командного рядка, що забезпечує взаємодію користувача з Docker. CLI надає інструментарій для створення, запуску, зупинки, видалення та моніторингу контейнерів і образів. Він підтримує широкий набір команд, які дозволяють ефективно керувати всіма компонентами екосистеми Docker без використання графічного інтерфейсу (рис. 2.12).

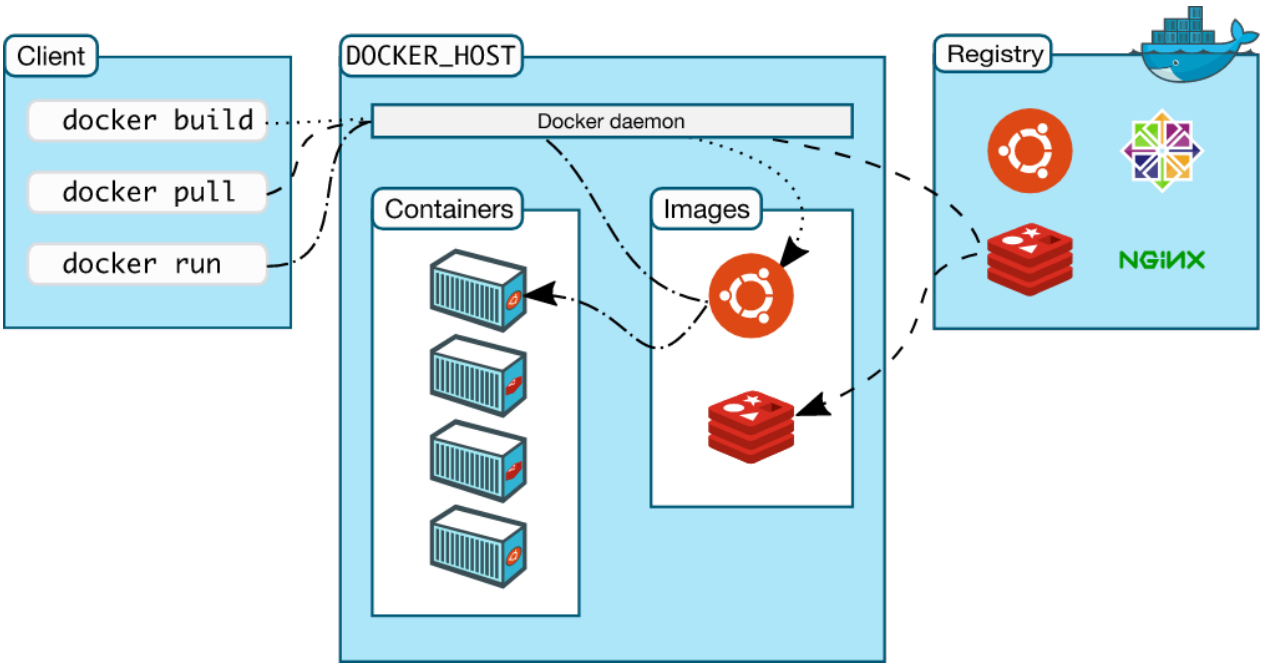


Рисунок 2.12 – Docker CLI

— Docker Swarm (режим Swarm у Docker Engine) — це вбудований інструмент Docker для організації кластерного середовища, яке дозволяє керувати декількома Docker-хостами як єдиною логічною системою. Використовуючи Swarm, можна створювати високодоступні, масштабовані розподілені застосунки, автоматизувати балансування навантаження та забезпечити безперервну роботу сервісів у разі відмови окремих вузлів.

— Docker Compose — інструмент для опису та координації запуску багатьох взаємопов'язаних контейнерів у межах одного застосунку. Compose спрощує процес розгортання, дозволяючи за допомогою одного конфігураційного файлу (docker-compose.yml) визначити всі сервіси, які входять до складу програми, а також їхню взаємодію.

Завдяки поєднанню таких інструментів, як Docker Engine, Compose та Swarm, платформа Docker забезпечує швидку розробку, зручне розгортання, ізоляцію середовища виконання та легке масштабування. Саме ці характеристики обумовлюють її популярність серед розробників програмного забезпечення, системних адміністраторів та DevOps-інженерів.

2.4 Розробка Telegram-бота з використанням бібліотеки Telegraf

Telegraf (рис 2.13) — це популярна JavaScript/Node.js бібліотека для створення Telegram-ботів. Вона надає простий і зручний API для взаємодії з Telegram Bot API. Telegraf дозволяє обробляти повідомлення, кнопки, команди, відповіді на інлайн-запити, взаємодію з Webhook та багато іншого[4].

```
bot.on("callback_query", (callbackQuery) => {
    const message = callbackQuery.message;

    if(callbackQuery.data == 'order'){
        askLocation(message);
    } else if(callbackQuery.data == 'aboutUs') {
        bot.sendMessage(message.chat.id,'Added to category \"' + callbackQuery.data + '\"!');
    } else if(callbackQuery.data == 'contact'){
        bot.sendMessage(message.chat.id,'Added new website, but '+ callbackQuery.data + 'there was no OG data!');
    } else{
        bot.editMessageText(message.chat.id, message.message_id, callbackQuery.inline_message_id , 'test',{
            reply_markup: {
                inline_keyboard: [[
                  {
                    text: 'Done',
                    callback_data:'done'
                  }
                ]]
            }
         });
        bot.sendMessage(message.chat.id,'failed'+message.message_id+' '+callbackQuery.inline_message_id);
    }
});
```

Рисунок 2.13 – Приклад використання Telegraf

Основні особливості Telegraf:

— Простий синтаксис, код на Telegraf легко читати й писати навіть новачкам.

— Підтримка всіх можливостей Telegram Bot API. Наприклад: інлайн-кнопки, клавіатури, опитування, медіа, геолокації, запит контактів тощо.

— Сумісність із TypeScript. Для великих проєктів корисна строгість і типізація.

— Гнучка система middlewares. Можна додавати проміжні обробники для авторизації, логування, обробки помилок тощо.

— Взаємодія з базами даних (наприклад, MongoDB, PostgreSQL).

— Можна легко інтегрувати з іншими API або сервісами (наприклад, Google Translate, Weather API тощо).

3 СТВОРЕННЯ TELEGRAM-БОТА ДЛЯ ВИВЧЕННЯ ІНОЗЕМНИХ СЛІВ

3.1 Реєстрація Telegram-бота та отримання токена доступу

Щоб створити Telegram-бота, спочатку потрібно звернутись до BotFather і виконати наступні дії[5]:

1. Відкрити Telegram та знайти користувача @BotFather (рис.3.1, це офіційний бот для створення інших ботів)[14].

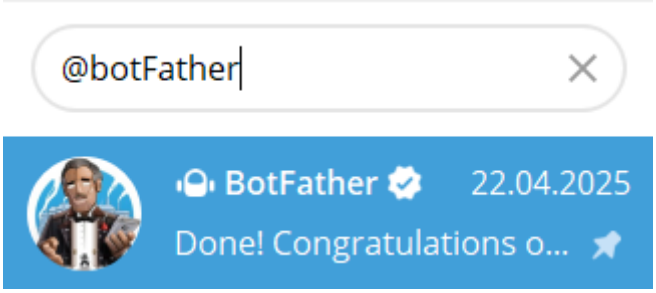


Рисунок 3.1 – Користувача @BotFather

2. Натиснути "Start" або ввести команду /start (рис.3.2), щоб активувати взаємодію.

Рисунок 3.2 – Реакція @BotFather на команду /start

3. Ввести команду /newbot (рис.3.3).

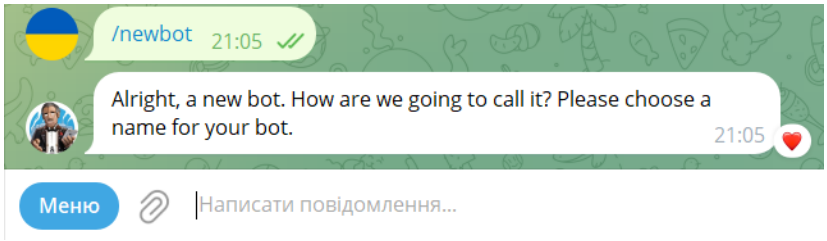


Рисунок 3.3 – Реакція @BotFather на команду / newbot

4. Вказати назву бота (будь-яке ім’я, наприклад, “DUIKT”) та ім’я користувача бота (username), яке має закінчуватись на bot (наприклад, DUIKTBot рис.3.4).

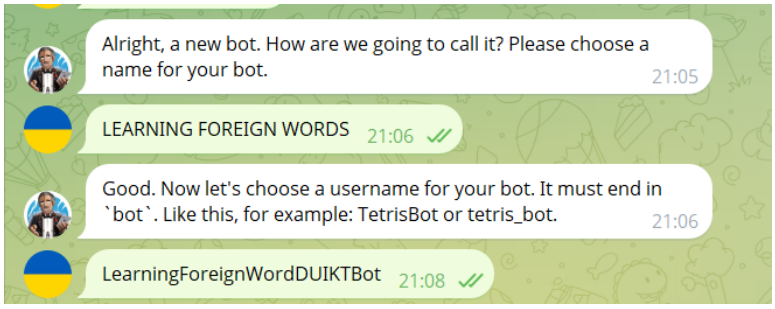


Рисунок 3.4 – Заповнення назви та ім’я користувача бота

5. Після цього BotFather надішле вам токен доступу — унікальний ключ, який дозволяє програмі керувати ботом (рис.3.5) через Telegram API.

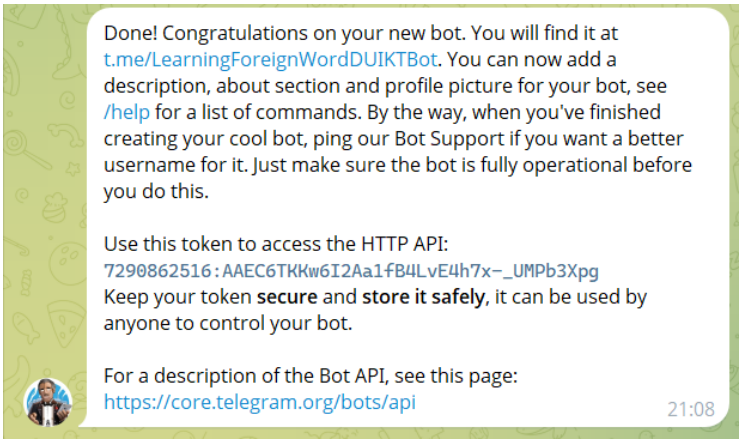


Рисунок 3.5 – Унікальний ключ, який дозволяє програмі керувати ботом

3.2 Структура проекту та демонстрація роботи

Основна мета Telegram-бота – допомогти користувачам вивчати іноземні слова за допомогою системи інтервального повторення. Для реалізації було обрано мову програмування JavaScript з використанням середовища Node.js та бібліотеки Telegraf для роботи з Telegram API. Також використано базу даних MongoDB для зберігання слів та інформації про користувача.

Для зберігання слів і всієї пов'язаної з ними інформації використовується MongoDB. Створено схему wordSchema (рис.3.6), яка описує структуру документа у базі. Вона містить поля для самого слова, перекладу, користувача, статистики правильних/неправильних відповідей, рівня знань та інтервалів повторення:

```
const mongoose = require("mongoose");

// Створюємо схему для слів
const wordSchema = new mongoose.Schema({
  // ID користувача, якому належить слово
  userId: {
    type: Number,
    required: true,
    index: true,
  },
  // Слово
  word: {
    type: String,
    required: true,
  },
  // Переклад слова
  translation: {
    type: String,
    required: true,
  },
  // Кількість правильних відповідей
  correctAnswers: {
    type: Number,
    default: 0,
  },
  // Кількість правильних відповідей
  correctAnswers: {
    type: Number,
    default: 0,
  },
  // Кількість неправильних відповідей
  incorrectAnswers: {
    type: Number,
    default: 0,
  },
  // Рівень знання слова (0-5)
  knowledgeLevel: {
    type: Number,
    default: 0,
  },
  // Наступна дата повторення
  nextReviewDate: {
    type: Date,
    default: Date.now,
  },
  // Інтервал повторення в днях
  reviewInterval: {
    type: Number,
    default: 1,
  },
  // Дата створення
  createdAt: {
    type: Date,
    default: Date.now,
  },
});
```

Рисунок 3.6 – Схема wordSchema

Особливу увагу заслуговує метод updateKnowledgeLevel (рис.3.7), який автоматично оновлює знання користувача про слово на основі правильності відповіді. Це дозволяє реалізувати алгоритм інтервального повторення.

```
// Метод для оновлення рівня знання та інтервалу повторення
wordSchema.methods.updateKnowledgeLevel = function (isCorrect) {
  if (isCorrect) {
    this.correctAnswers += 1;
    if (this.knowledgeLevel < 5) {
      this.knowledgeLevel += 1;
    }
    this.reviewInterval = Math.max(
      1,
      Math.floor(Math.pow(1.5, this.knowledgeLevel))
    );
  } else {
    this.incorrectAnswers += 1;
    if (this.knowledgeLevel > 0) {
      this.knowledgeLevel -= 1;
    }
    this.reviewInterval = 1;
  }

  // Встановлюємо наступну дату повторення
  this.nextReviewDate = new Date(
    Date.now() + this.reviewInterval * 24 * 60 * 60 * 1000
  );
};
```

Рисунок 3.7 – Метод updateKnowledgeLevel

Підключення до бази даних відбувається через бібліотеку mongoose (рис.3.8), яка забезпечує зручну взаємодію з MongoDB.

```
mongoose
  .connect(process.env.MONGODB_URI)
  .then(() => console.log("Підключено до MongoDB"))
  .catch((err) => console.error("Помилка підключення:", err));
```

Рисунок 3.8 – Підключення до бази даних через бібліотеку mongoose

Використання змінних середовища через файл .env забезпечує гнучкість та безпеку зберігання токенів.

Telegram-бот створюється за допомогою конструктора Telegraf (рис.3.9). Для збереження стану між повідомленнями використовується сесія.

```
const bot = new Telegraf(process.env.TELEGRAM_BOT_TOKEN
bot.use(
  session({
    defaultSession: () => ({
      awaitingWord: false,
      awaitingAnswer: false,
      currentWord: null,
      showWord: false,
      isPractice: false,
      currentPage: 0,
      words: [],
    }),
  })
```

Рисунок 3.9 – Конструктор Telegraf

Це дозволяє зберігати тимчасову інформацію, як-от: чи чекає бот на введення слова, чи відповідає користувач на запитання тощо.

Для взаємодії з користувачем створюється інлайн-клавіатура з основними командами (рис.3.10 – 3.11).

```
js

const mainKeyboard = Markup.inlineKeyboard([
  [
    Markup.button.callback("➕ Додати слово", "add_word"),
    Markup.button.callback("📋 Список слів", "word_list"),
  ],
  [
    Markup.button.callback("🎯 Практика", "practice"),
    Markup.button.callback("📊 Статистика", "stats"),
  ],
]);
```

Рисунок 3.10 – Інлайн-клавіатура з основними командами в коді

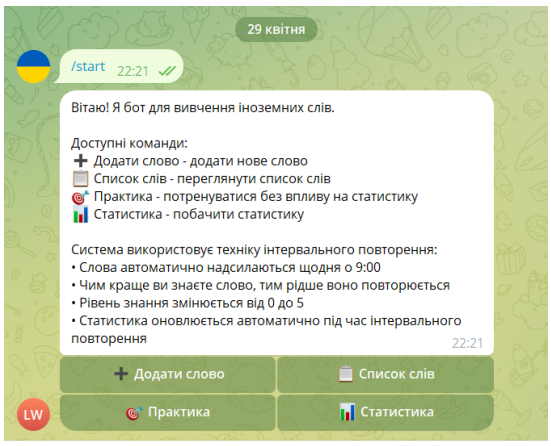


Рисунок 3.11 – Інлайн-клавіатура з основними командами в боті

Після натискання на одну з кнопок, бот переходить у відповідний режим.

Коли користувач натискає "Додати слово", бот запитує слово та переклад у форматі "word переклад"(рис.3.12 – 3.13).

```
bot.action("add_word", (ctx) => {
  ctx.session.awaitingWord = true;
  ctx.reply(
    "Введіть слово та його переклад у форматі:\nhello привіт",
    Markup.inlineKeyboard([[Markup.button.callback("⬅ Назад", "back")]])
  );
});
```

Рисунок 3.12 – Функція додати слово в коді

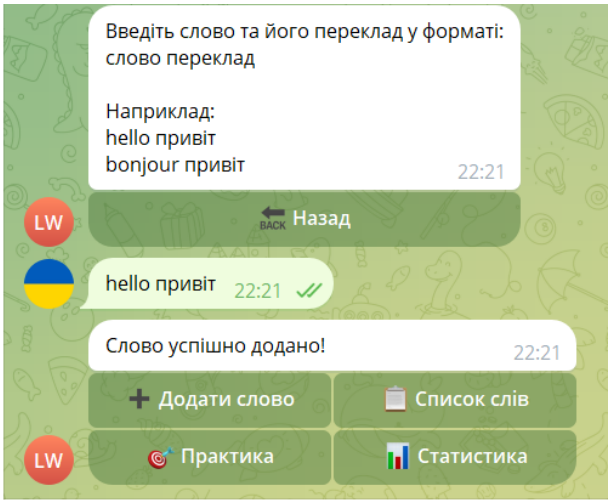


Рисунок 3.13 – Функція додати слово в боті

У відповідь на текстове повідомлення, бот розділяє його на частини, зберігає у базі даних і надсилає підтвердження.

Користувач може переглянути список своїх слів (рис.3.14 – 3.15), які зберігаються у базі даних. Для зручності список розбивається на сторінки по 10 слів.

```
function showWordList(ctx, page) {
  const wordsPerPage = 10;
  const currentWords = ctx.session.words.slice(page * wordsPerPage, (page + 1) * wordsPerPage);
  // Формується повідомлення з перекладами
}
```

Рисунок 3.14 – Функція переглянути список своїх слів в коді

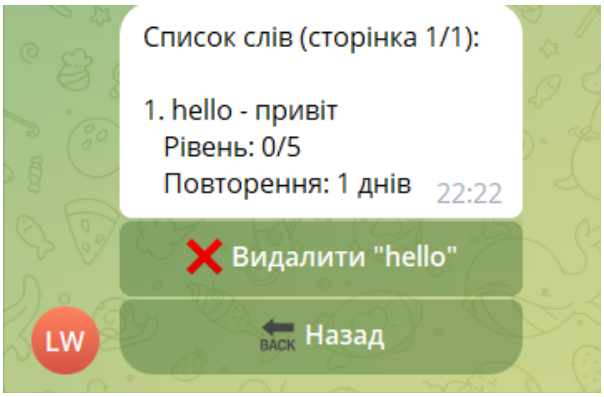


Рисунок 3.15 – Функція переглянути список своїх слів в боті

Кожне слово містить інформацію про рівень знань (від 0 до 5) та інтервал повторення.

Користувач може видалити слово зі списку, натиснувши кнопку "Видалити"(рис.3.16– 3.17). Для цього бот використовує ID документа у базі.

```
bot.action(/delete_(.+)/, async (ctx) => {
  const wordId = ctx.match[1];
  await Word.findByIdAndDelete(wordId);
  // Оновлюється список
});
```

Рисунок 3.16 – Функція видалити в коді

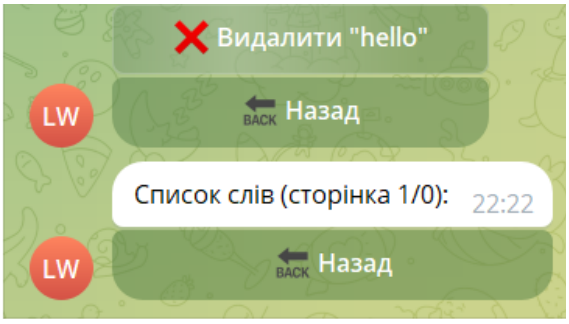


Рисунок 3.17 – Функція видалити в боті

Головною особливістю бота є автоматичне повторення слів за технікою інтервального повторення (рис.3.18 – 3.19). Слова надсилаються щодня, з урахуванням рівня знань.

```
bot.action("get_word", async (ctx) => {
  const word = await Word.findOne({
    userId: ctx.from.id,
    nextReviewDate: { $lte: new Date() },
  }).sort({ knowledgeLevel: 1 });
});
```

Рисунок 3.18 – Функція інтервальне повторення в коді

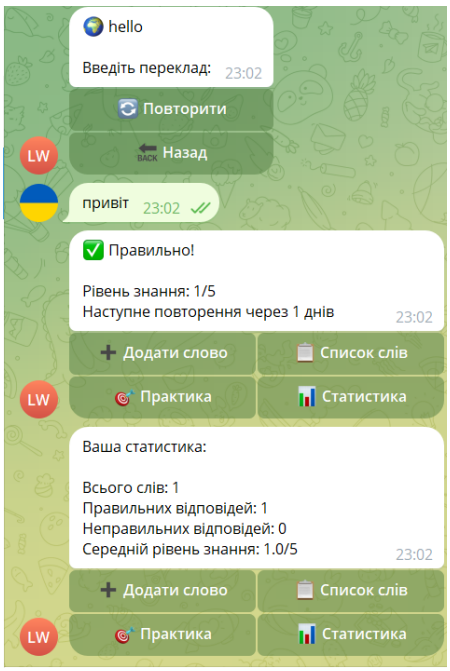


Рисунок 3.19 – Функція інтервальне повторення в боті

Якщо користувач правильно відповідає — рівень слова підвищується, а дата наступного повторення зміщується далі.

Окрім основного навчання, реалізовано режим практики (рис.3.20 – 3.21). У цьому режимі статистика не змінюється. Це зручно для простої перевірки знань.

```
bot.action("practice", async (ctx) => {
  const randomIndex = Math.floor(Math.random() * totalWords);
  const word = await Word.findOne({ userId: ctx.from.id }).skip(randomIndex);
  ctx.session.isPractice = true;
});
```

Рисунок 3.20 – Функція практики в коді

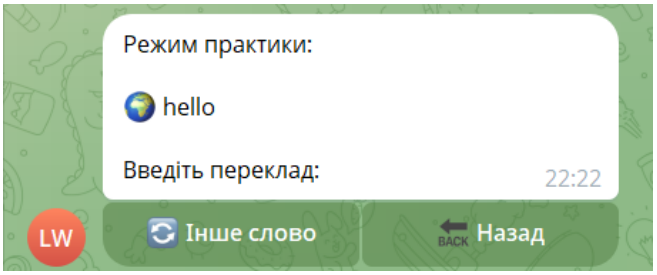


Рисунок 3.21 – Функція практики в боті

Коли користувач вводить переклад, бот перевіряє правильність і, за необхідності, оновлює рівень знань (рис.3.22 – 3.23).

```
if (isCorrect) {
  ctx.reply("✅ Правильно!", mainKeyboard);
} else {
  ctx.reply(`❌ Неправильно. Правильна відповідь: ${correctAnswer}`);
}
```

Рисунок 3.22 – Реакція на відповідь в боті

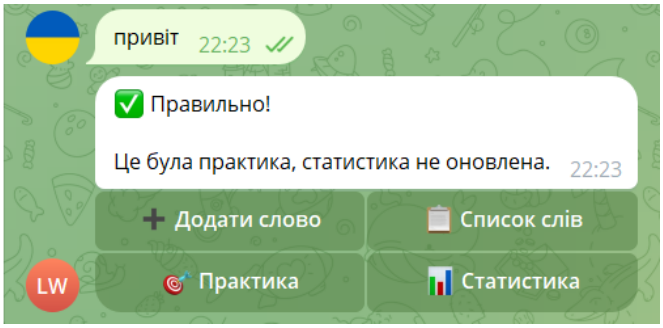


Рисунок 3.23 – Реакція на відповідь в боті

ВИСНОВКИ

Telegram-боти стали невід'ємною частиною цифрового середовища, завдяки своїй простоті, доступності та зручності. У процесі виконання дипломної роботи було досліджено можливості використання чат-ботів для організації навчального процесу, зокрема — вивчення іноземних слів. Порівняння мобільних застосунків і чат-ботів дозволило визначити переваги останніх у контексті швидкого старту, мінімального інтерфейсу та кращої інтеграції у повсякденну комунікацію користувача.

Було проведено огляд типів чат-ботів, особливостей їх реалізації та прикладів уже існуючих рішень у сфері мовного навчання. На основі цього аналізу обґрунтовано доцільність створення власного Telegram-бота як альтернативи складним мобільним застосункам.

Для реалізації функціоналу було обрано сучасний технологічний стек: Node.js як серверне середовище виконання, MongoDB як база даних для збереження слів та статистики користувача, Docker — для контейнеризації та полегшеного розгортання застосунку, а також бібліотеку Telegraf — як основний інструмент взаємодії з Telegram API. Завдяки цим інструментам було реалізовано повноцінного Telegram-бота з можливістю додавання власних слів, інтервального повторення, тренування та статистичного відстеження прогресу.

Особливу увагу приділено інтервальному повторенню як ефективній методиці запам'ятовування. Реалізований механізм динамічного оновлення рівня знань дозволяє адаптувати навчальний процес під кожного користувача.

Висновки, отримані в ході роботи, свідчать про актуальність використання Telegram-ботів у сфері самостійного навчання. Розроблений прототип підтверджує ефективність такого підходу та може бути основою для подальшого розширення функціоналу, включаючи підтримку інших мов, інтеграцію з API перекладу, голосове введення та використання елементів штучного інтелекту.

Подальші інвестиції у розвиток подібних ботів є доцільними, оскільки вони дозволяють демократизувати доступ до освіти, зробити навчання більш персоналізованим і доступним будь-де, будь-коли — лише через звичний месенджер.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is Node.js? | Node.js. Node.js Foundation. URL: https://nodejs.org/en/about (дата звернення: 29.04.2025).
2. Getting Started with MongoDB. MongoDB Documentation. URL: https://www.mongodb.com/docs/manual/introduction (дата звернення: 29.04.2025).
3. Introduction to Mongoose for MongoDB. Mongoose Documentation. URL: https://mongoosejs.com/docs/index.html (дата звернення: 29.04.2025).
4. Telegraf.js – Telegram Bot Framework for Node.js. GitHub. URL: https://github.com/telegraf/telegraf (дата звернення: 29.04.2025).
5. Telegram Bot API. Telegram. URL: https://core.telegram.org/bots/api (дата звернення: 29.04.2025).
6. How to Create a Telegram Bot with Node.js. DigitalOcean Community. URL: https://www.digitalocean.com/community/tutorials/how-to-build-a-telegram-bot-with-node-js (дата звернення: 29.04.2025).
7. Docker Overview | Docker Docs. Docker. URL: https://docs.docker.com/get-started/overview (дата звернення: 29.04.2025).
8. Learn JavaScript. Mozilla Developer Network. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 29.04.2025).
9. JavaScript Asynchronous Programming: Promises and Async/Await. JavaScript.info. URL: https://javascript.info/async (дата звернення: 29.04.2025).
10. Environment Variables in Node.js. Node.js Documentation. URL: https://nodejs.dev/en/learn/how-to-read-environment-variables-from-nodejs (дата звернення: 29.04.2025).
11. What Is Spaced Repetition? | Anki Manual. Anki. URL: https://docs.ankiweb.net/intro.html#what-is-spaced-repetition (дата звернення: 29.04.2025).

12. RepeatWordBot – Telegram Bot for Language Learning. Telegram Bot Directory. URL: https://t.me/RepeatWordBot (дата звернення: 29.04.2025).

13. Quizlet Features and Benefits. Quizlet. URL: https://quizlet.com/features (дата звернення: 29.04.2025).

14. Creating a Bot with BotFather. Telegram Help Center. URL: https://core.telegram.org/bots#botfather (дата звернення: 29.04.2025).

15. REST API Design Best Practices. IBM Developer. URL: https://developer.ibm.com/articles/rest-api-design-best-practices (дата звернення: 29.04.2025).

КОД TELEGRAM-БОТУ

```
// Імпортуємо необхідні бібліотеки
const { Telegraf, session, Markup } =
require("telegraf");
const mongoose = require("mongoose");
const cron = require("node-cron");
require("dotenv").config();

// Імпортуємо модель Word
const Word = require("./models/Word");

// Підключаємося до MongoDB
mongoose
  .connect(process.env.MONGODB_URI)
  .then(() => console.log("Підключено до
MongoDB"))
  .catch((err) => console.error("Помилка
підключення до MongoDB:", err));

// Створюємо екземпляр бота
const bot = new
Telegraf(process.env.TELEGRAM_BOT_TOK
EN);

// Використовуємо сесії з початковим станом
bot.use(
  session({
    defaultSession: () => ({
      awaitingWord: false,
      awaitingAnswer: false,
      currentWord: null,
      showWord: false,
      isPractice: false,
      currentPage: 0,
      words: [],
    }),
  })
);

// Створюємо клавіатуру з основними
командами
const mainKeyboard =
Markup.inlineKeyboard([
  [
    Markup.button.callback("➕ Додати слово",
"add_word"),
    Markup.button.callback("📋 Список слів",
"word_list"),
  ],
  [
    Markup.button.callback("🎯 Практика",
"practice"),
    Markup.button.callback("📊 Статистика",
"stats"),
  ],
  //[Markup.button.callback("❓ Допомога",
"help")],
]);

// Створюємо сховище для стану cron job
const cronState = new Map();
// Обробник команди /start
bot.command("start", (ctx) => {
  ctx.reply(
```

```
    "Вітаю! Я бот для вивчення іноземних слів.\n\n" +
      "Доступні команди:\n" +
      "➕ Додати слово - додати нове слово\n" +
      "📋 Список слів - переглянути список слів\n" +
      "🎯 Практика - потренуватися без впливу на статистику\n" +
      "📊 Статистика - побачити статистику\n\n" +
      "Система використовує техніку інтервального повторення:\n" +
      "• Слова автоматично надсилаються щодня о 9:00\n" +
      "• Чим краще ви знаєте слово, тим рідше воно повторюється\n" +
      "• Рівень знання змінюється від 0 до 5\n" +
      "• Статистика оновлюється автоматично під час інтервального повторення",
    mainKeyboard
  );
});

// Обробник для додавання слова
bot.action("add_word", (ctx) => {
  ctx.session.awaitingWord = true;
  ctx.reply(
    "Введіть слово та його переклад у форматі:\n" +
      "слово переклад\n\n" +
      "Наприклад:\n" +
      "hello привіт\n" +
      "bonjour привіт",
    Markup.inlineKeyboard([[Markup.button.callback("🔙 Назад", "back")]])
  );
});

// Обробник для перегляду списку слів
bot.action("word_list", async (ctx) => {
  try {
    const words = await Word.find({ userId: ctx.from.id }).sort({ word: 1 });

    if (words.length === 0) {
      ctx.reply("У вас ще немає доданих слів.", mainKeyboard);
      return;
    }

    // Розділяємо слова на сторінки по 10 слів
    const wordsPerPage = 10;
    const totalPages = Math.ceil(words.length / wordsPerPage);
    ctx.session.currentPage = 0;
    ctx.session.words = words;

    showWordList(ctx, 0);
  } catch (error) {
    console.error("Помилка при отриманні списку слів:", error);
    ctx.reply("Сталася помилка при отриманні списку слів.", mainKeyboard);
  }
});
```

```
// Функція для відображення списку слів
function showWordList(ctx, page) {
  const words = ctx.session.words;
  const wordsPerPage = 10;
  const start = page * wordsPerPage;
  const end = Math.min(start + wordsPerPage, words.length);
  const currentWords = words.slice(start, end);

  let message = `Список слів (сторінка ${page + 1}/${Math.ceil(
    words.length / wordsPerPage
  )}):\n\n`;

  currentWords.forEach((word, index) => {
    message += `${start + index + 1}. ${word.word} - ${word.translation}\n`;
    message += `   Рівень: ${word.knowledgeLevel}/5\n`;
    message += `   Повторення: ${word.reviewInterval} днів\n\n`;
  });

  const keyboard = [];
  const row = [];

  // Додаємо кнопки для навігації
  if (page > 0) {
    row.push(Markup.button.callback("⬅️ Попередня", "prev_page"));
  }
  if (end < words.length) {
    row.push(Markup.button.callback("Наступна ➡️", "next_page"));
  }
  if (row.length > 0) {
    keyboard.push(row);
  }

  // Додаємо кнопки для видалення слів
  currentWords.forEach((word, index) => {
    keyboard.push([
      Markup.button.callback(
        `❌ Видалити "${word.word}"`,
        `delete_${word._id}`
      ),
    ]);
  });

  keyboard.push([Markup.button.callback("🔙 Назад", "back")]);

  ctx.reply(message, Markup.inlineKeyboard(keyboard));
}

// Обробники для навігації по списку слів
bot.action("prev_page", (ctx) => {
  ctx.session.currentPage--;
  showWordList(ctx, ctx.session.currentPage);
});

bot.action("next_page", (ctx) => {
  ctx.session.currentPage++;
  showWordList(ctx, ctx.session.currentPage);
});
```

```
// Обробник для видалення слова
bot.action(/delete_(.+)/, async (ctx) => {
  try {
    const wordId = ctx.match[1];
    await Word.findByIdAndDelete(wordId);

    // Оновлюємо список слів
    const words = await Word.find({ userId:
ctx.from.id }).sort({ word: 1 });
    ctx.session.words = words;

    // Перераховуємо поточну сторінку
    const wordsPerPage = 10;
    const totalPages = Math.ceil(words.length /
wordsPerPage);
    if (ctx.session.currentPage >= totalPages) {
      ctx.session.currentPage = Math.max(0,
totalPages - 1);
    }

    showWordList(ctx, ctx.session.currentPage);
  } catch (error) {
    console.error("Помилка при видаленні
слова:", error);
    ctx.reply("Сталася помилка при видаленні
слова.", mainKeyboard);
  }
});

// Обробник для отримання слова
bot.action("get_word", async (ctx) => {
  try {
    // Спочатку шукаємо слово для
повторення за розкладом
    let word = await Word.findOne({
      userId: ctx.from.id,
      nextReviewDate: { $lte: new Date() },
    }).sort({ knowledgeLevel: 1,
nextReviewDate: 1 });

    // Якщо немає слів для повторення за
розкладом, беремо випадкове слово
    if (!word) {
      const totalWords = await
Word.countDocuments({ userId: ctx.from.id });
      if (totalWords > 0) {
        const randomIndex =
Math.floor(Math.random() * totalWords);
        word = await Word.findOne({ userId:
ctx.from.id }).skip(randomIndex);
      }
    }

    if (word) {
      ctx.session.currentWord = word;
      ctx.session.awaitingAnswer = true;
      ctx.session.isPractice = false; //
Позначаємо, що це не режим практики
      // Випадково вибираємо, показувати
слово чи переклад
      const showWord = Math.random() < 0.5;
      ctx.session.showWord = showWord;

      ctx.reply(
        `Слово для вивчення:\n\n` +
        `🌍 ${showWord ? word.word :
word.translation}\n\n` +
```

```
        `Введіть ${showWord ? "переклад" : "слово"}:`,
        Markup.inlineKeyboard([
         [
          Markup.button.callback("🔄 Повторити", "repeat"),
          Markup.button.callback("🎲 Інше слово", "another_word"),
         ],
         [Markup.button.callback("🔙 Назад", "back")],
        ])
      );
    } else {
      ctx.reply(
        "На жаль, у вас ще немає доданих слів. Додайте слова для початку навчання.",
        mainKeyboard
      );
    }
  } catch (error) {
    console.error("Помилка при отриманні слова:", error);
    ctx.reply("Сталася помилка. Спробуйте пізніше.", mainKeyboard);
  }
});

// Обробник для режиму практики
bot.action("practice", async (ctx) => {
  try {
    const totalWords = await Word.countDocuments({ userId: ctx.from.id });
    if (totalWords > 0) {
      const randomIndex = Math.floor(Math.random() * totalWords);
      const word = await Word.findOne({ userId: ctx.from.id }).skip(
        randomIndex
      );

      ctx.session.currentWord = word;
      const showWord = Math.random() < 0.5;
      ctx.session.showWord = showWord;
      ctx.session.awaitingAnswer = true;
      ctx.session.isPractice = true; // Позначаємо, що це режим практики

      ctx.reply(
        `Режим практики:\n\n` +
          `🌍 ${showWord ? word.word : word.translation}\n\n` +
          `Введіть ${showWord ? "переклад" : "слово"}:`,
        Markup.inlineKeyboard([
          [
            Markup.button.callback("🔄 Інше слово", "practice"),
            Markup.button.callback("🔙 Назад", "back"),
          ],
        ])
      );
    } else {
      ctx.reply("У вас ще немає доданих слів.", mainKeyboard);
    }
  } catch (error) {
```

```
    console.error("Помилка при отриманні слова для практики:", error);
    ctx.reply("Сталася помилка. Спробуйте пізніше.", mainKeyboard);
  }
});

// Додаємо cron job для інтервального повторення
cron.schedule("* * * * *", async () => {
  try {
    // Знаходимо всіх користувачів
    const users = await Word.distinct("userId");

    for (const userId of users) {
      // Знаходимо слова для повторення
      const wordsToReview = await Word.find({
        userId,
        nextReviewDate: { $lte: new Date() },
      });

      if (wordsToReview.length > 0) {
        for (const word of wordsToReview) {
          const showWord = Math.random() < 0.5;
          const message =
            `🌍 ${showWord ? word.word : word.translation}\n\n` +
            `Введіть ${showWord ? "переклад" : "слово"}:`;

          try {
            await bot.telegram.sendMessage(
              userId,
              message,
              Markup.inlineKeyboard([
                [Markup.button.callback("🔄 Повторити", "repeat")],
                [Markup.button.callback("🔙 Назад", "back")],
              ])
            );

            // Зберігаємо стан для цього користувача
            cronState.set(userId, {
              awaitingAnswer: true,
              currentWord: word,
              showWord: showWord,
              isPractice: false,
            });
          } catch (error) {
            console.error(
              `Помилка при відправці слова користувачу ${userId}:`,
              error
            );
          }
        }
      }
    }
  } catch (error) {
    console.error("Помилка в cron job:", error);
  }
});

// Модифікуємо обробник текстових повідомлень
bot.on("text", async (ctx) => {
  // Якщо очікуємо додавання слова
```

```
  if (ctx.session.awaitingWord) {
    const parts = ctx.message.text.split(" ");
    if (parts.length >= 2) {
      try {
        const word = parts[0];
        const translation = parts.slice(1).join(" ");

        const newWord = new Word({
          userId: ctx.from.id,
          word,
          translation,
        });

        await newWord.save();
        ctx.session.awaitingWord = false;
        ctx.reply("Слово успішно додано!",
mainKeyboard);
      } catch (error) {
        console.error("Помилка при додаванні
слова:", error);
        ctx.reply("Помилка при додаванні слова.
Спробуйте ще раз.");
      }
    } else {
      ctx.reply(
        "Неправильний формат. Введіть слово
та його переклад через пробіл."
      );
    }
  }
  // Якщо очікуємо відповідь на слово
  else if (
    (ctx.session.awaitingAnswer &&
ctx.session.currentWord) ||
    (cronState.has(ctx.from.id) &&
cronState.get(ctx.from.id).awaitingAnswer)
  ) {
    const state = ctx.session.awaitingAnswer
      ? ctx.session
      : cronState.get(ctx.from.id);
    const userAnswer =
ctx.message.text.toLowerCase();
    const correctAnswer = state.showWord
      ?
state.currentWord.translation.toLowerCase()
      : state.currentWord.word.toLowerCase();

    const isCorrect = userAnswer ===
correctAnswer;

    // Оновлюємо статистику тільки якщо це
не режим практики
    if (!state.isPractice) {

state.currentWord.updateKnowledgeLevel(isCo
rrect);
      await state.currentWord.save();
    }

    if (isCorrect) {
      ctx.reply(
        "✅ Правильно!\n\n" +
          (state.isPractice
            ? "Це була практика, статистика не
оновлена."
            : `Рівень знання:
${state.currentWord.knowledgeLevel}/5\n` +
              `Наступне повторення через
${state.currentWord.reviewInterval} днів`),
```

```
      mainKeyboard
    );
  } else {
    ctx.reply(
      `❌ Неправильно. Правильна відповідь: ${correctAnswer}\n\n` +
        (state.isPractice
          ? "Це була практика, статистика не оновлена."
          : `Рівень знання: ${state.currentWord.knowledgeLevel}/5\n` +
            `Наступне повторення через ${state.currentWord.reviewInterval} днів`),
      mainKeyboard
    );
  }

  // Очищаємо стан
  if (ctx.session.awaitingAnswer) {
    ctx.session.awaitingAnswer = false;
    ctx.session.currentWord = null;
    ctx.session.showWord = false;
    ctx.session.isPractice = false;
  }
  if (cronState.has(ctx.from.id)) {
    cronState.delete(ctx.from.id);
  }
 }
});

// Обробник для кнопки repeat
bot.action("repeat", async (ctx) => {
 if (!ctx.session.currentWord) {
  ctx.reply(
    "Слово не знайдено. Спробуйте отримати нове слово.",
    mainKeyboard
  );
  return;
 }

 const word = ctx.session.currentWord;
 const showWord = Math.random() < 0.5;
 ctx.session.showWord = showWord;
 ctx.session.awaitingAnswer = true;

 ctx.reply(
  `Слово для вивчення:\n\n` +
   `🌍 ${showWord ? word.word : word.translation}\n\n` +
   `Введіть ${showWord ? "переклад" : "слово"}:`,
  Markup.inlineKeyboard([
   [
    Markup.button.callback("🔄 Повторити", "repeat"),
    Markup.button.callback("🎲 Інше слово", "another_word"),
   ],
   [Markup.button.callback("🔙 Назад", "back")],
  ])
 );
});

// Обробник для кнопки another_word
bot.action("another_word", async (ctx) => {
 try {
```

```
    const totalWords = await
Word.countDocuments({ userId: ctx.from.id });
    if (totalWords > 0) {
      const randomIndex =
Math.floor(Math.random() * totalWords);
      const word = await Word.findOne({ userId:
ctx.from.id }).skip(
        randomIndex
      );

      ctx.session.currentWord = word;
      const showWord = Math.random() < 0.5;
      ctx.session.showWord = showWord;
      ctx.session.awaitingAnswer = true;

      ctx.reply(
        `Слово для вивчення:\n\n` +
         `🌍 ${showWord ? word.word :
word.translation}\n\n` +
         `Введіть ${showWord ? "переклад" :
"слово"}:`,
        Markup.inlineKeyboard([
          [
            Markup.button.callback("🔄
Повторити", "repeat"),
            Markup.button.callback("🎲 Інше
слово", "another_word"),
          ],
          [Markup.button.callback("🔙 Назад",
"back")],
        ])
      );
    } else {
      ctx.reply("У вас ще немає доданих слів.",
mainKeyboard);
    }
  } catch (error) {
    console.error("Помилка при отриманні
випадкового слова:", error);
    ctx.reply("Сталася помилка. Спробуйте
пізніше.", mainKeyboard);
  }
});

// Обробник для кнопки back
bot.action("back", (ctx) => {
  ctx.session.awaitingWord = false;
  ctx.session.awaitingAnswer = false;
  ctx.session.currentWord = null;
  ctx.reply("Повертаємось до головного
меню", mainKeyboard);
});

// Обробник для статистики
bot.action("stats", async (ctx) => {
  try {
    const stats = await Word.aggregate([
      {
        $match: {
          userId: ctx.from.id,
        },
      },
      {
        $group: {
          _id: null,
          totalWords: { $sum: 1 },
          totalCorrect: { $sum: "$correctAnswers"
},
```

```
        totalIncorrect: { $sum:
"$incorrectAnswers" },
        averageLevel: { $avg:
"$knowledgeLevel" },
      },
    },
  ]);

  if (stats.length === 0) {
    ctx.reply("У вас ще немає доданих слів.",
mainKeyboard);
    return;
  }

  const stat = stats[0];
  let message = "Ваша статистика:\n\n";
  message +=
    `Всього слів: ${stat.totalWords}\n` +
    `Правильних відповідей:
${stat.totalCorrect}\n` +
    `Неправильних відповідей:
${stat.totalIncorrect}\n` +
    `Середній рівень знання:
${stat.averageLevel.toFixed(1)}/5\n\n`;

  ctx.reply(message, mainKeyboard);
 } catch (error) {
  console.error("Помилка при отриманні
статистики:", error);
  ctx.reply("Сталася помилка при отриманні
статистики.", mainKeyboard);
 }
});

// Обробник для допомоги
bot.action("help", (ctx) => {
 ctx.reply(
  "Щоб отримати детальну інформацію,
використайте команду /start",
  mainKeyboard
 );
});

// Запускаємо бота
bot.launch().then(() => {
 console.log("Бот запущено");
});

// Обробка помилок
bot.catch((err, ctx) => {
 console.error("Помилка в боті:", err);
 ctx.reply("Сталася помилка. Спробуйте
пізніше.");
});

// Обробка завершення роботи
process.once("SIGINT", () =>
bot.stop("SIGINT"));
process.once("SIGTERM", () =>
bot.stop("SIGTERM"));
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота на ступінь бакалавра за темою:

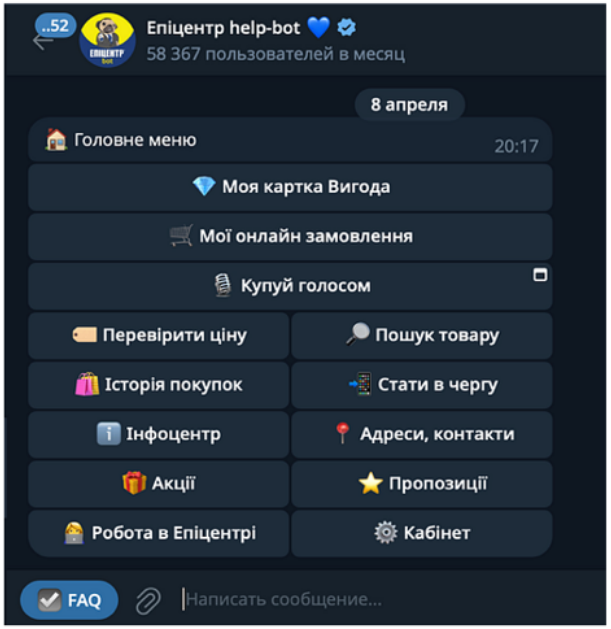
«TELEGRAM-БОТ ДЛЯ ВИВЧЕННЯ ІНОЗЕМНИХ СЛІВ З ВИКОРИСТАННЯМ JAVASCRIPT ТА NODE.JS »

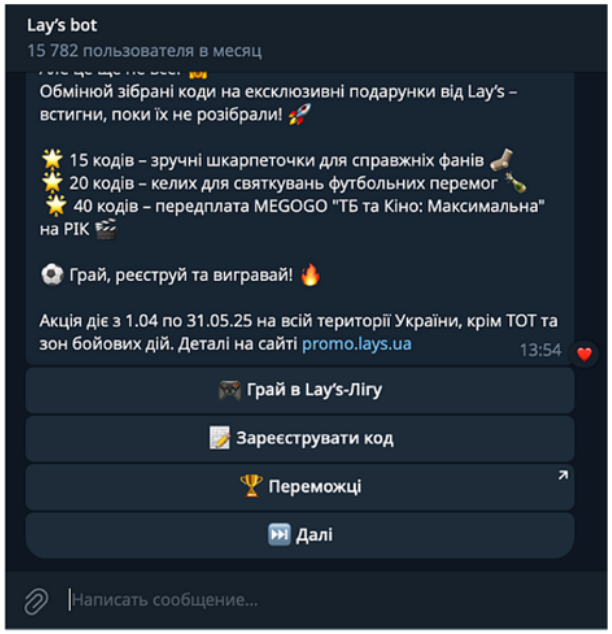
Виконала:
студентка групи КНД-42
Марія Лозенко
Керівник:
доцент кафедри комп'ютерних наук, доктор філософії
Березовська Юлія

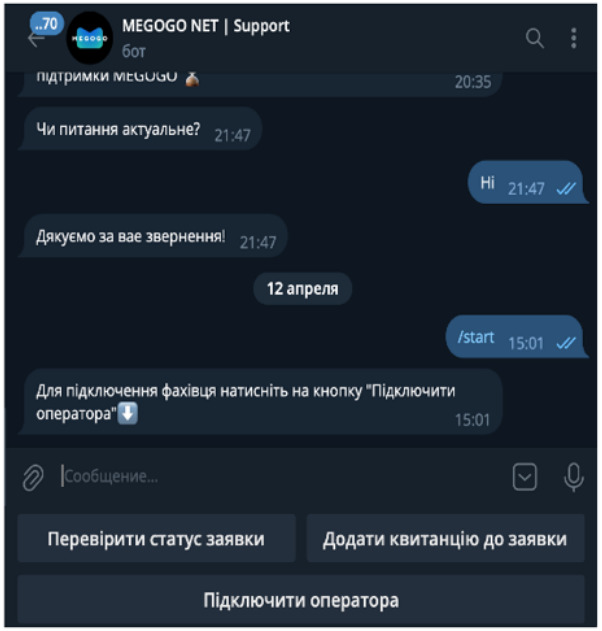
ЗАГАЛЬНА ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

Мета дослідження:	розробка Telegram-бота для вивчення іноземних слів з використанням JavaScript та Node.js.
Об’єкт дослідження:	інформаційна система для вивчення іноземної лексики у форматі чат-бота.
Предмет дослідження:	telegram-бот як засіб організації процесу вивчення іноземних слів.

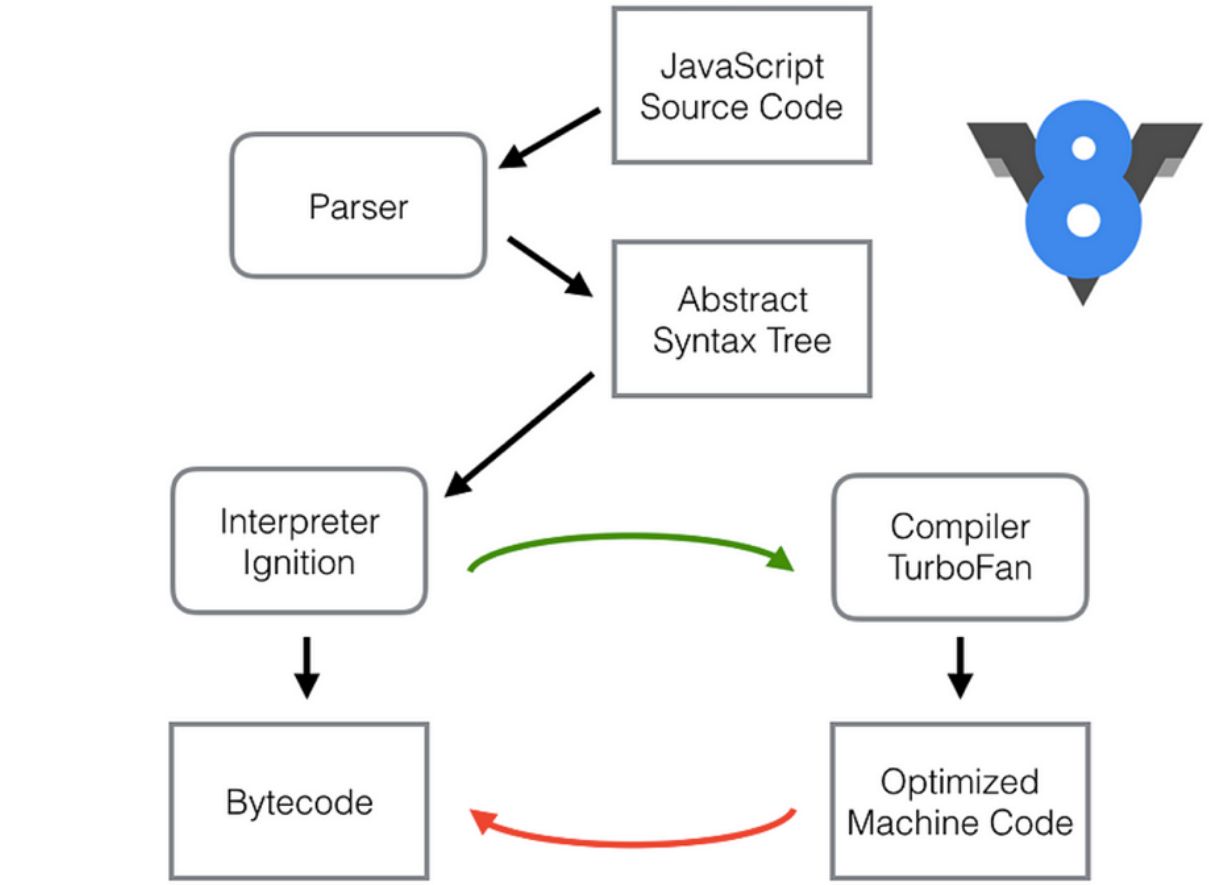
ТЕЛЕГРАМ-БОТ







NODE.JS


JavaScript
Source Code
Parser
Abstract
Syntax Tree
Interpreter
Ignition
Compiler
TurboFan
Bytecode
Optimized
Machine Code
Google
@fhinkel

TELEGRAF

```
// Імпортуємо необхідні бібліотеки
const { Telegraf, session, Markup } = require("telegraf");


// Обробник команди /start
bot.command("start", (ctx) => {
  ctx.reply(
    "Вітаю! Я бот для вивчення іноземних слів.\n\n" +
      "Доступні команди:\n" +
      "➕ Додати слово – додати нове слово\n" +
      "📋 Список слів – переглянути список слів\n" +
      "🎯 Практика – потренуватися без впливу на статистику\n" +
      "📊 Статистика – побачити статистику\n\n" +
      "Система використовує техніку інтервального повторення:\n" +
      "• Слова автоматично надсилаються щодня о 9:00\n" +
      "• Чим краще ви знаєте слово, тим рідше воно повторюється\n" +
      "• Рівень знання змінюється від 0 до 5\n" +
      "• Статистика оновлюється автоматично під час інтервального повторення",
    mainKeyboard
  );
});

// Обробник для додавання слова
bot.action("add_word", (ctx) => {
  ctx.session.awaitingWord = true;
  ctx.reply(
    "Введіть слово та його переклад у форматі:\n" +
      "слово переклад\n\n" +
      "Наприклад:\n" +
      "hello привіт\n" +
      "bonjour привіт",
    Markup.inlineKeyboard([[Markup.button.callback("⬅️ Назад", "back")]])
  );
});

// Обробник для перегляду списку слів
bot.action("word_list", async (ctx) => {
  try {
    const words = await Word.find({ userId: ctx.from.id }).sort({ word: 1 });

    if (words.length === 0) {
      ctx.reply("У вас ще немає доданих слів.", mainKeyboard);
      return;
```

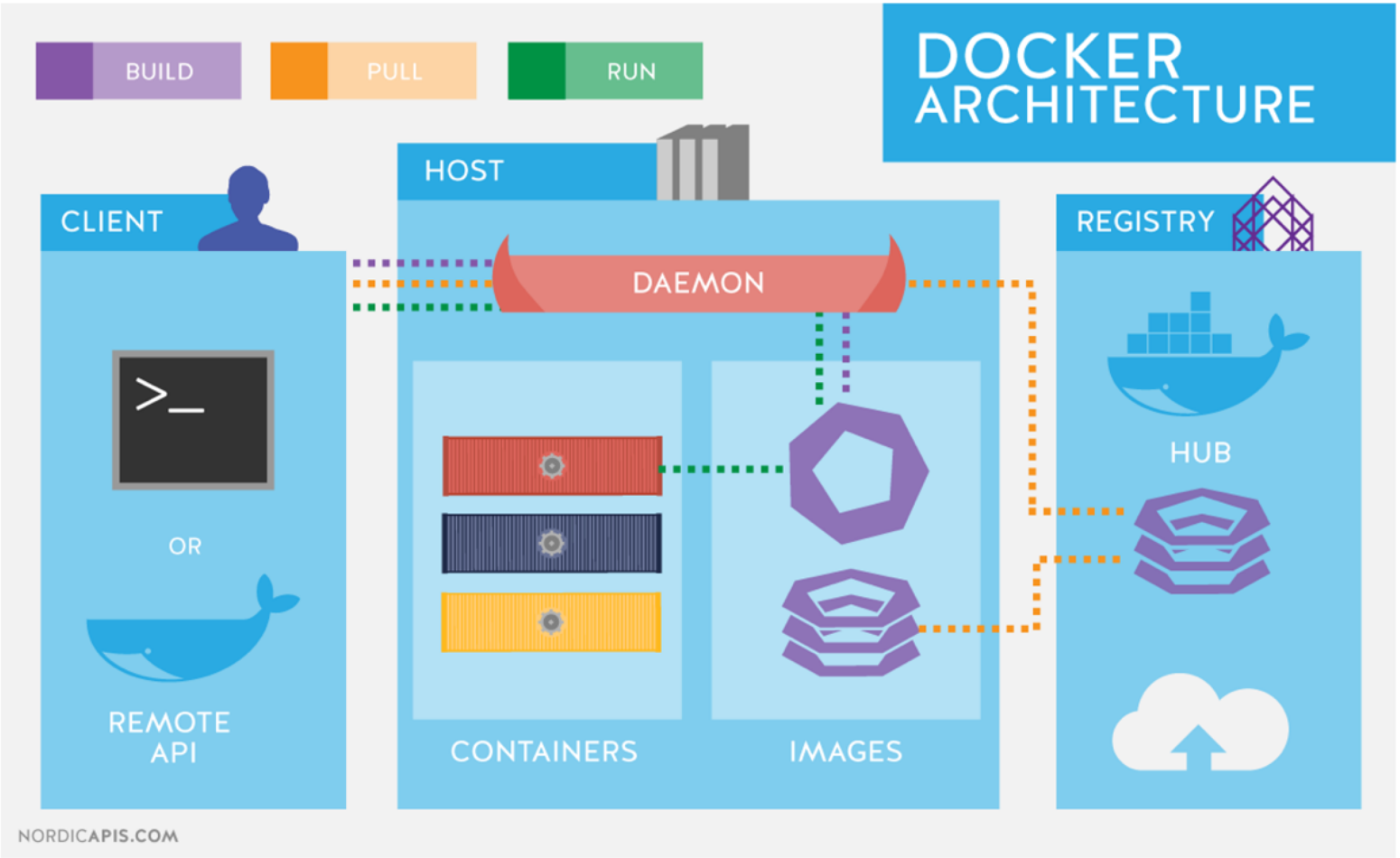
MongoDB

```
{
        _id : <ObjectId> ,

        CustomerName : Guru99 ,

        Order:
                {

                }
}
```

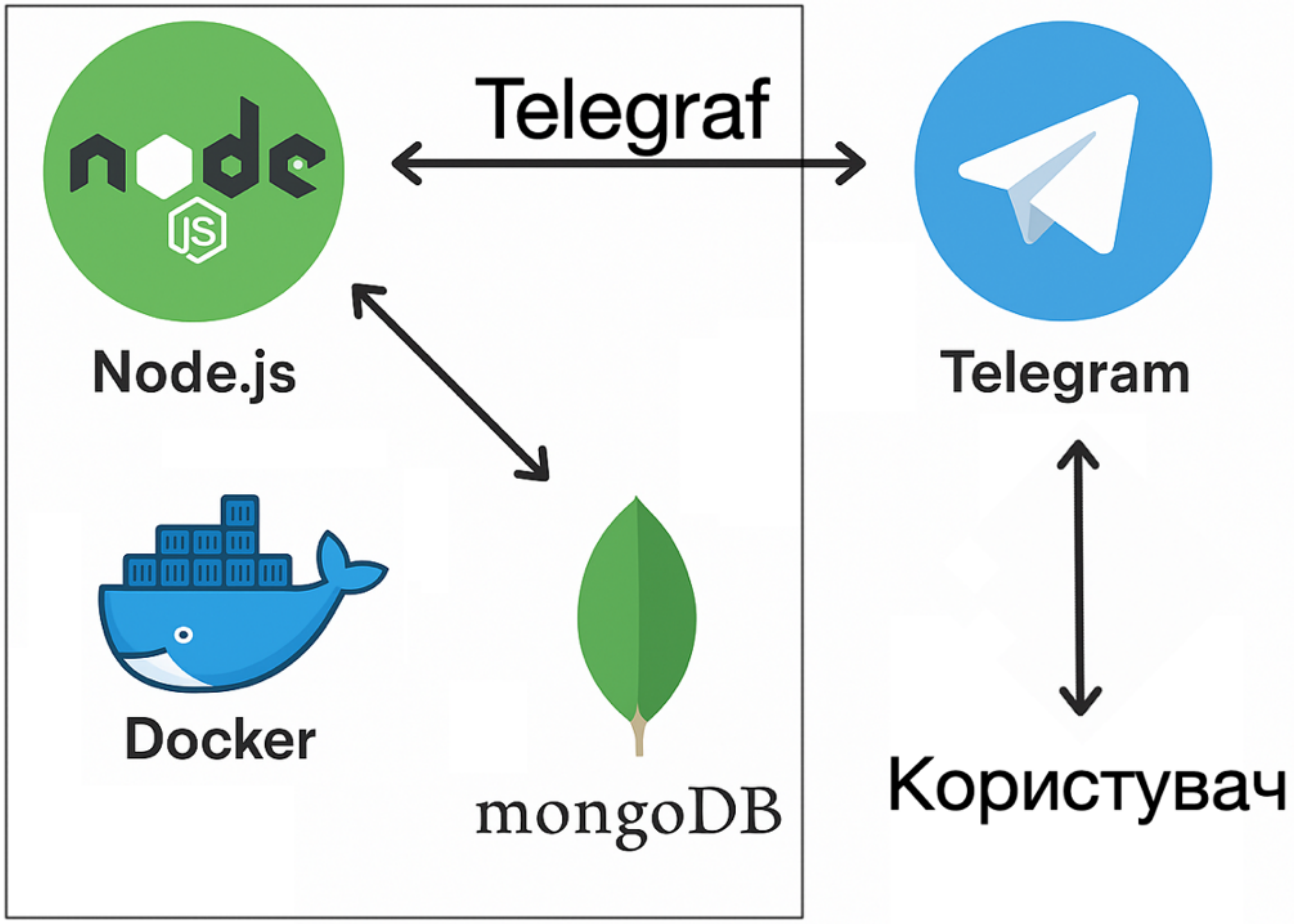
```
OrderID: 111
Product: ProductA
Quantity: 5
```

Example of how data can be embedded in a document

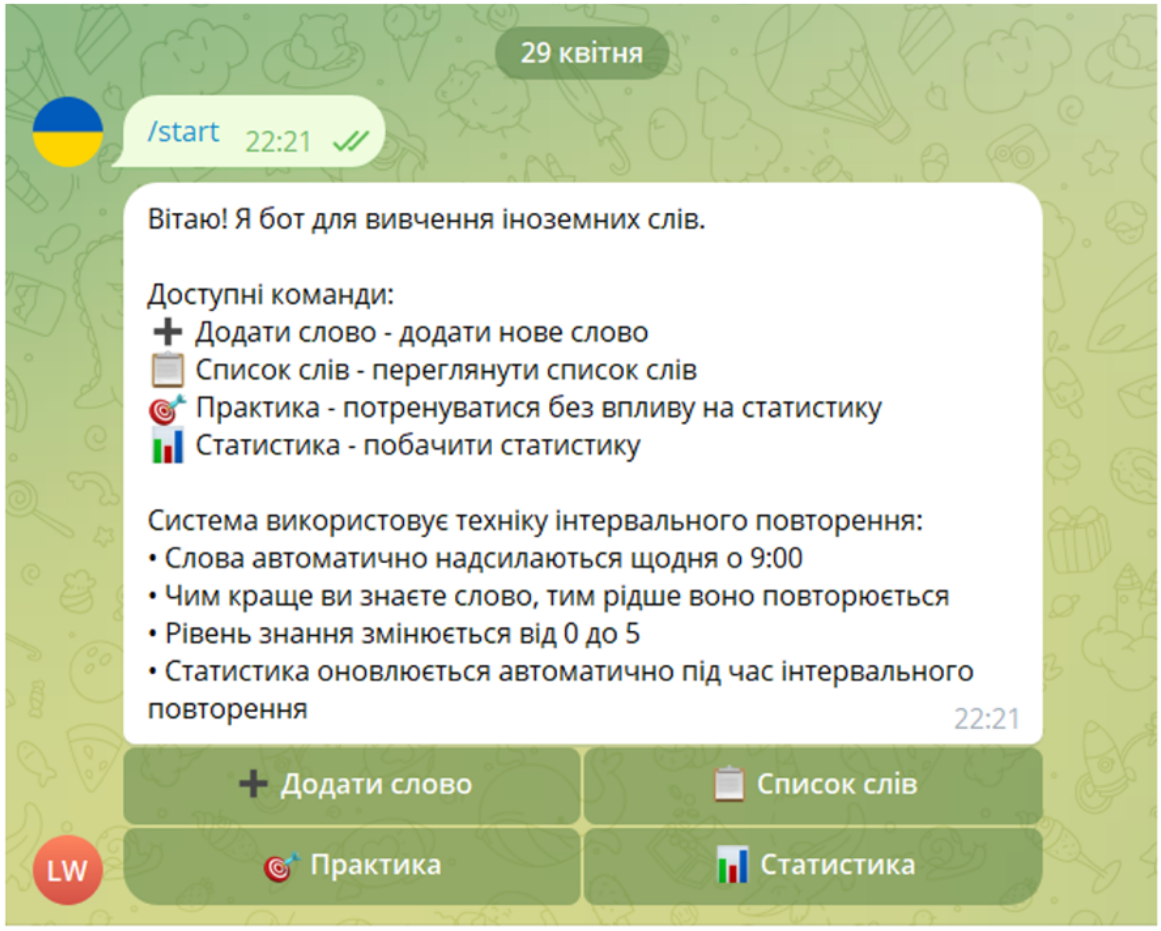
Docker


BUILD
PULL
RUN
DOCKER ARCHITECTURE
HOST
CLIENT
DAEMON
REGISTRY
>_
OR
HUB
REMOTE API
CONTAINERS
IMAGES
NORDICAPIS.COM

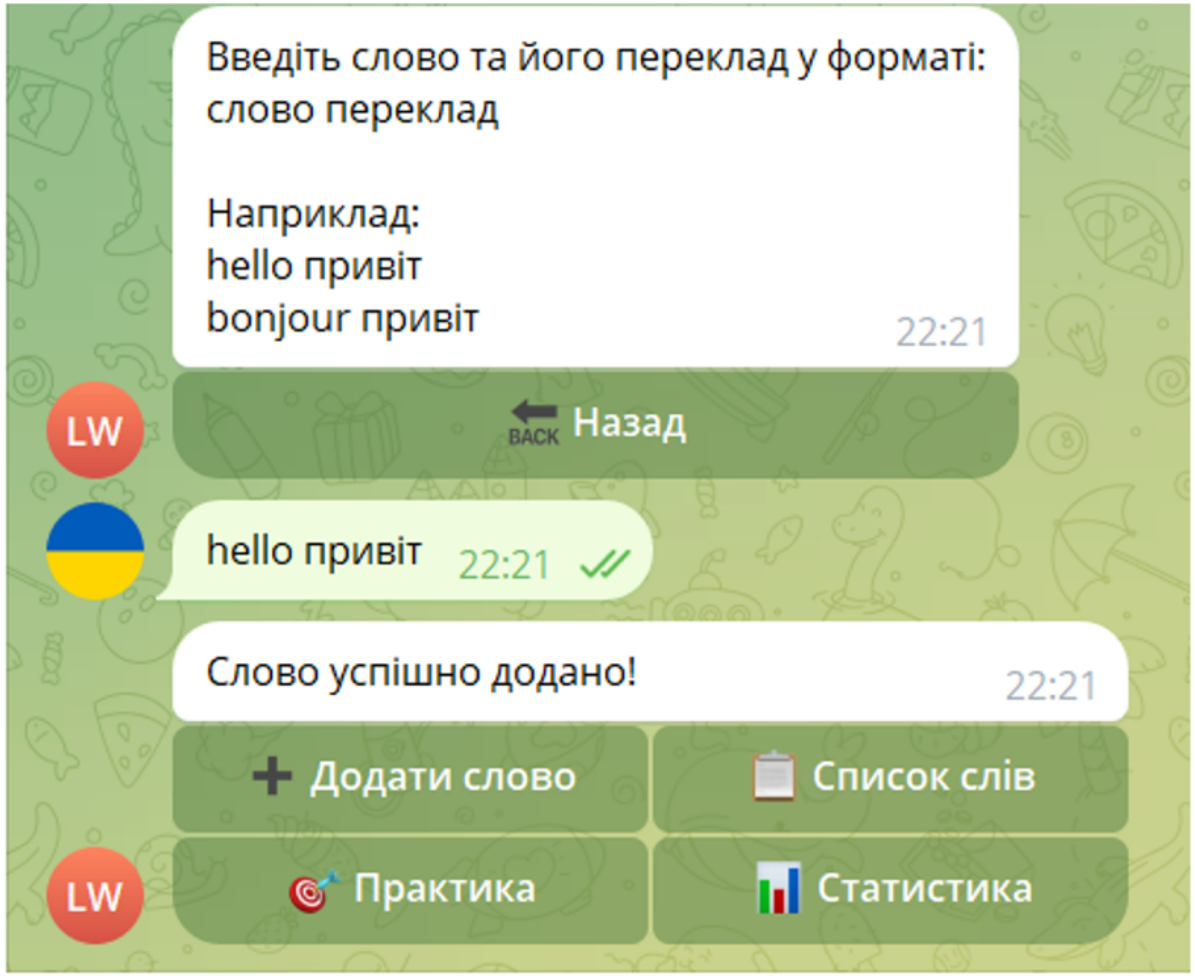
ЗАГАЛЬНА СХЕМА РОБОТИ



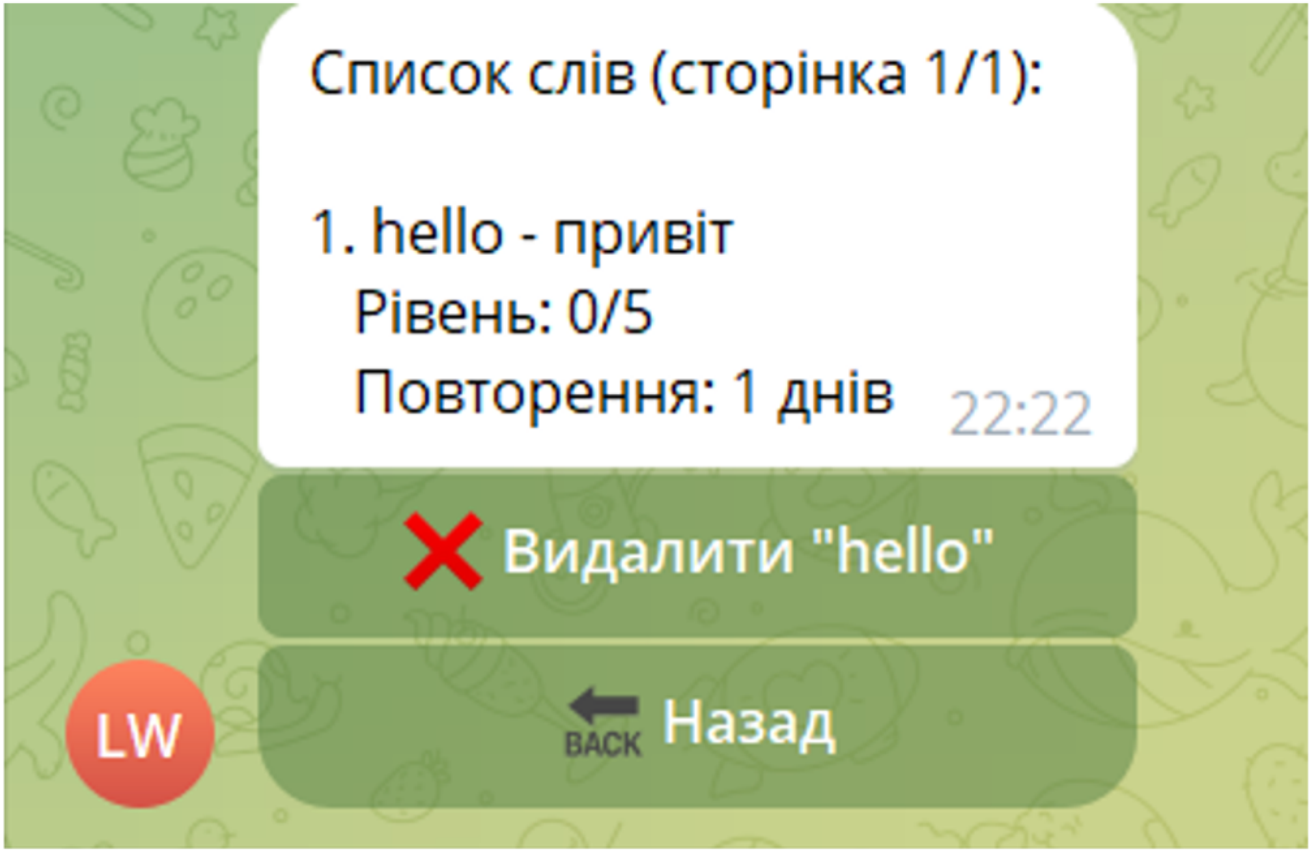
ДЕМОНСТРАЦІЯ РОБОТИ



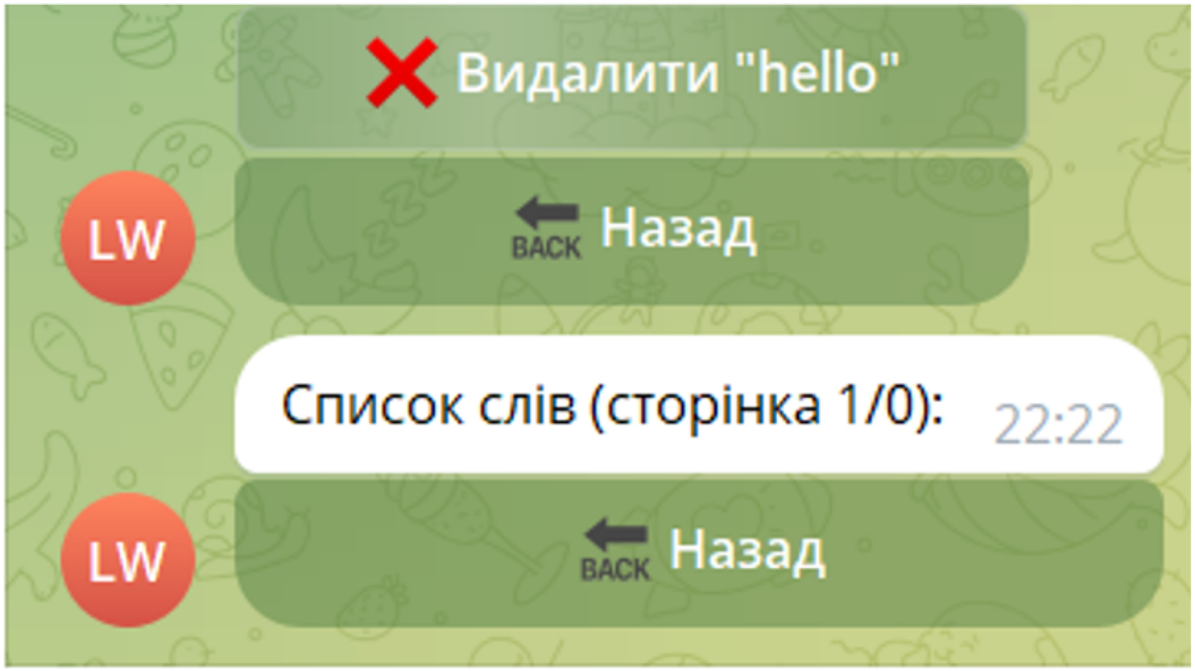
ФУНКЦІЯ ДОДАТИ СЛОВО В БОТІ



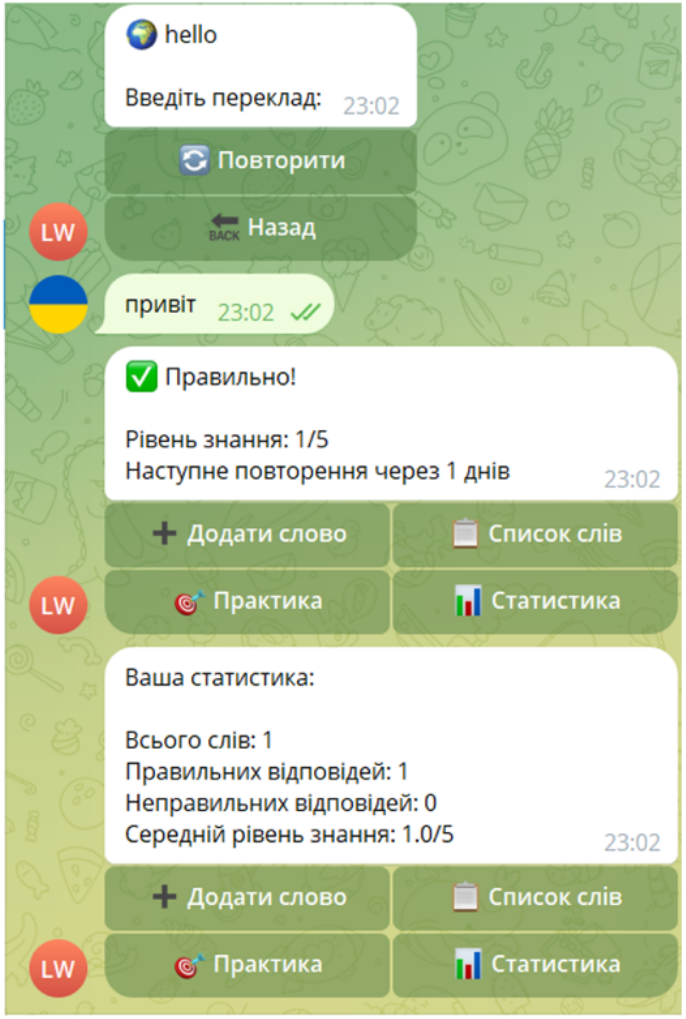
ФУНКЦІЯ ПЕРЕГЛЯНУТИ СПИСОК СВОЇХ СЛІВ В БОТІ



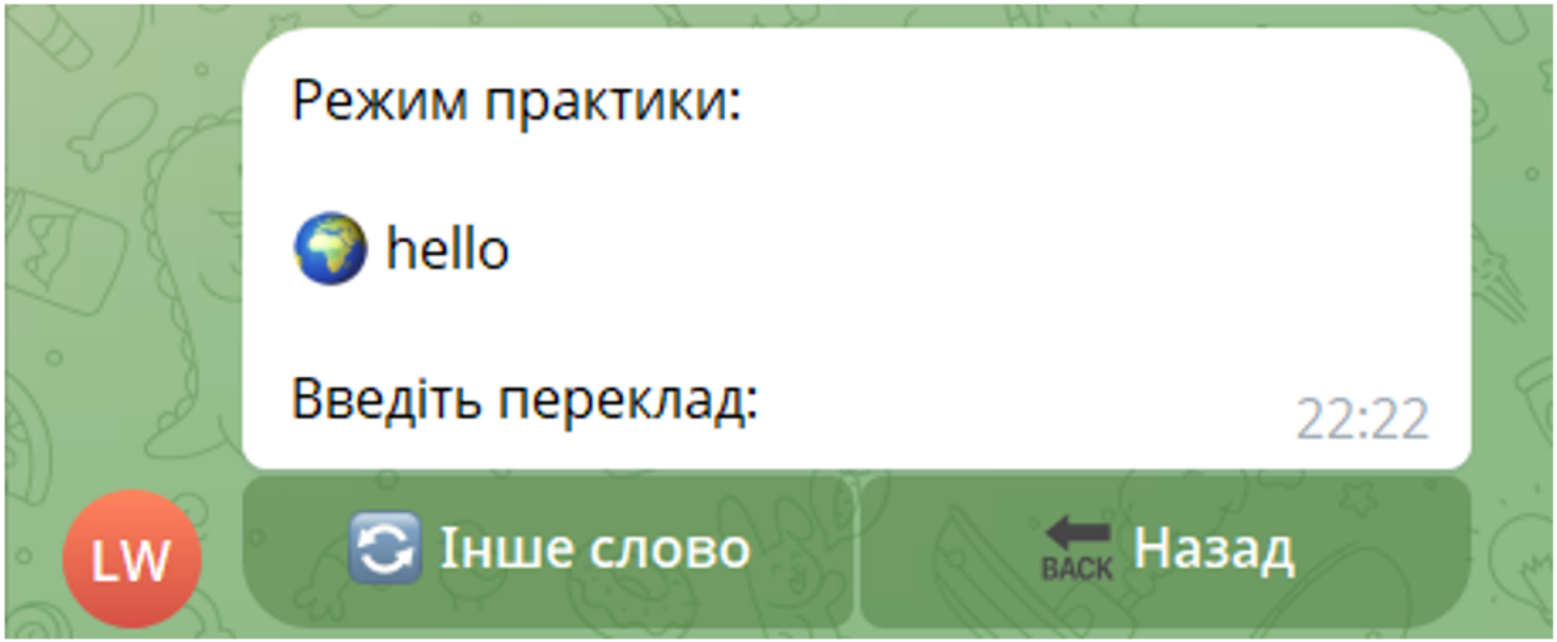
ФУНКЦІЯ ВИДАЛИТИ В БОТІ



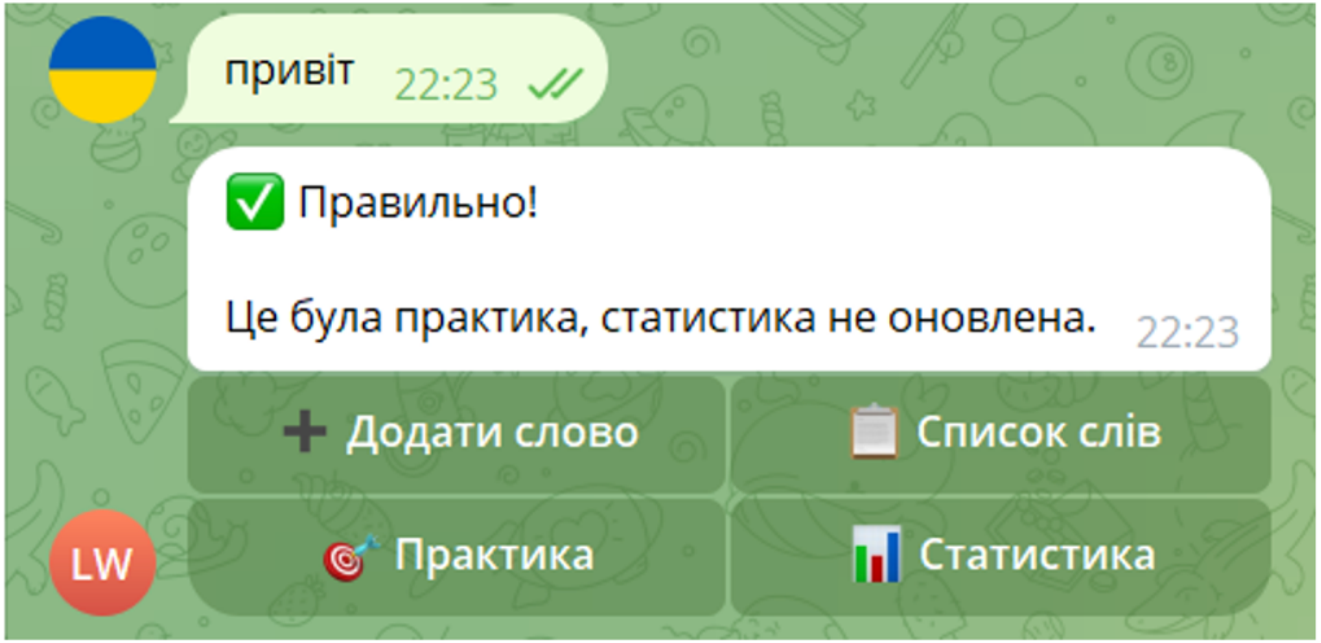
ФУНКЦІЯ ІНТЕРВАЛЬНЕ ПОВТОРЕННЯ В БОТІ



ФУНКЦІЯ ПРАКТИКИ В БОТІ



РЕАКЦІЯ НА ВІДПОВІДЬ В БОТІ



ВИСНОВКИ

- Було проведено огляд типів чат-ботів, особливостей їх реалізації та прикладів уже існуючих рішень у сфері мовного навчання. На основі цього аналізу обґрунтовано доцільність створення власного Telegram-бота як альтернативи складним мобільним застосункам.
- Для реалізації функціоналу було обрано сучасний технологічний стек: Node.js як серверне середовище виконання, MongoDB як база даних для збереження слів та статистики користувача, Docker — для контейнеризації та полегшеного розгортання застосунку, а також бібліотеку Telegraf — як основний інструмент взаємодії з Telegram API. Завдяки цим інструментам було реалізовано повноцінного Telegram-бота з можливістю додавання власних слів, інтервального повторення, тренування та статистичного відстеження прогресу.
- Особливу увагу приділено інтервальному повторенню як ефективній методиці запам’ятовування. Реалізований механізм динамічного оновлення рівня знань дозволяє адаптувати навчальний процес під кожного користувача.
- Висновки, отримані в ході роботи, свідчать про актуальність використання Telegram-ботів у сфері самостійного навчання. Розроблений прототип підтверджує ефективність такого підходу та може бути основою для подальшого розширення функціоналу, включаючи підтримку інших мов, інтеграцію з API перекладу, голосове введення та використання елементів штучного інтелекту.