МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ САДІВНИЦТВА
Факультет економіки і підприємництва
Кафедра інформаційних технологій

Допущено до захисту
"_____" _____ 2024 р.
Зав. кафедри, к.т.н., доцент
_____ **Ліщук Р.І.**

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПОШУКУ АВТОМОБІЛІВ У ПРОДАЖУ

Виконав: студент 2 курсу, 21 м-кн групи
Освітньо-професійної програми
«Комп'ютерні науки»
спеціальності 122 «Комп'ютерні науки»
Новіцький Павло Олександрович

Науковий керівник:
д.т.н., професор
Кучерук Володимир Юрійович

Умань – 2024

ЗМІСТ

ВСТУП 3
РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ 5
1.1. Аналіз сучасного стану та актуальності проблеми. 5
1.2. Огляд існуючих підходів до автоматизації пошуку автомобілів. 6
1.3. Аналіз API сервісів для роботи з інформацією про продаж автомобілів та розгляд ключових факторів 14
РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ ТА АРХІТЕКТУРА ПІДСИСТЕМИ 21
2.1. Формалізація задачі автоматизації пошуку автомобілів. 21
2.2. Обґрунтування вибору методів і інструментів для реалізації задачі. 23
2.3. Постановка технічних вимог до розробленої системи та визначення критеріїв оцінки якості 29
РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 33
3.1. Архітектура та алгоритми роботи телеграм-бота для пошуку автомобілів. 33
3.2. Взаємодія з API auto.ria.com: особливості реалізації. 38
3.3. Тестування системи та аналіз ефективності 40
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 50
Додаток А 54
Додаток Б 57

ВСТУП

На тлі стрімкого розвитку інформаційного суспільства автоматизація різноманітних процесів стала однією з головних вимог сучасного суспільства. Важливим напрямком, в якому інформаційні технології дозволяють значно підвищити ефективність, є пошук автомобілів для продажу. Ручний пошук транспортних засобів, що відповідають вимогам покупця, займає багато часу та сил. Використання автоматизованої системи пошуку спрощує цей процес і робить його максимально зручним для користувача.

В даний час існує велика кількість інтернет-ресурсів, які надають інформацію про продаж автомобілів, наприклад, популярна платформа auto.ria.com. Але проблема пошуку автомобілів з певними параметрами залишається актуальною для користувачів і часто ускладнюється великою кількістю доступної інформації. Інтеграція з API таких платформ за допомогою ботів Telegram дозволяє створити зручний інструмент, який швидко надає користувачам необхідну інформацію.

Актуальність теми. Сучасні інформаційні технології дозволяють розробляти інструменти для ефективної взаємодії з великими базами даних, спрощуючи процес пошуку, аналізу та фільтрації інформації. Розробка телеграм-бота для пошуку автомобілів на базі API платформи auto.ria.com є актуальною та практично важливою, оскільки дозволяє автоматизувати пошук автомобілів за заданими критеріями, що скорочує час та зусилля користувачів. . Така система має потенціал для розвитку в сенсі розширення її функціональності та адаптації до інших платформ.

Метою дослідження є розробка інформаційної технології у вигляді Telegram-бота для автоматизації пошуку автомобілів для продажу. Бот працюватиме з API платформи auto.ria.com і забезпечуватиме швидкий доступ до даних на основі заданих параметрів.

Для досягнення поставлених цілей необхідно вирішити наступні завдання:

1. Аналіз існуючих підходів до автоматизації пошуку автомобілів для продажу.
2. Вивчення можливостей та функціоналу API платформи auto.ria.com.
3. Дизайн та розробка Telegram-бота для взаємодії з API та реалізації пошукових функцій.
4. Тестування системи та аналіз її ефективності.
5. Формулювання рекомендацій щодо подальшого розвитку функціоналу Telegram-бота.

Метою дослідження є інформаційний процес автоматизації пошуку даних про продаж автомобілів.

Предмет дослідження – інструменти з API auto.ria.com, технології інтеграції API з ботами Telegram, методи пошуку та фільтрації інформації.

Методи дослідження. У дослідженні використовуються аналітичні та синтетичні методи дослідження інформаційно-пошукових технологій, методи проектування створення архітектури Telegram-ботів та емпіричні методи тестування розробленої системи.

База даних досліджень містить наукові дослідження з інформаційних технологій, технічну документацію API auto.ria.com, а також стандарти та рекомендації щодо створення Telegram-ботів.

Таким чином, розробка робота Telegram, який автоматизує пошук автомобілів на продаж, стане важливим кроком у вдосконаленні інформаційних технологій, спрямованих на підвищення зручності та ефективності взаємодії користувачів з великими базами даних.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз сучасного стану та актуальності проблеми.

У сучасному світі автомобілі — це більше, ніж просто засіб пересування; вони стали важливою частиною нашого способу життя. Попит на купівлю автомобілів продовжує зростати, тому з’являються нові платформи та сервіси для пошуку та продажу автомобілів. Одним із ключових аспектів для ефективної роботи такої платформи є автоматизація процесів. Це забезпечує швидкий доступ до потрібної інформації та спрощує взаємодію користувача з сервісом.

Незважаючи на розвиток цифрових технологій, багато користувачів стикаються з труднощами при пошуку автомобіля. Традиційні методи, такі як використання газетних оголошень або ручний перегляд численних сайтів, є застарілими та неефективними. У зв'язку з цим необхідно впроваджувати інформаційні технології, які оптимізують процес пошуку, роблять його зручним і швидким.

Однією з головних проблем є необхідність обробки великих обсягів даних для пошуку автомобілів на основі конкретних параметрів, таких як марка, модель, ціна, рік випуску та стан. Ключову роль у вирішенні цієї проблеми відіграють прикладні програмні інтерфейси (API), які надають доступ до баз даних платформ продажу автомобілів, таких як auto.ria.com, olx.ua та cars.com. API дозволяє розробникам інтегрувати функцію пошуку транспортних засобів у сторонні системи, роблячи інформацію легко доступною для кінцевих користувачів[9].

Важливість автоматизації процесу пошуку транспортних засобів зумовлена кількома важливими факторами:

1) Збільшення кількості автомобілів на ринку: виведення нових моделей і збільшення обсягів продажів вимагають швидкої та ефективної обробки інформації.
2) Зростаюча роль зручності для користувачів: у сучасному світі люди віддають перевагу службам, які дають їм швидкий доступ до необхідної інформації.

3) Розробка технологій штучного інтелекту: використання машинного навчання та алгоритмів аналізу даних значно покращує результати пошуку та робить їх більш релевантними.

Виходячи з аналізу поточної ситуації, можна зробити висновок, що автоматизація пошуку автомобілів не тільки відповідає сучасним тенденціям, але й є необхідною для розвитку ринку продажу автомобілів. Впровадження такого рішення дасть вашій платформі конкурентну перевагу та значно покращить користувацький досвід.

1.2. Огляд існуючих підходів до автоматизації пошуку автомобілів.

Автоматизація пошуку автомобілів є важливим завданням для розробників програмного забезпечення, які працюють з платформами електронної комерції, що спеціалізуються на продажу автомобілів. У цьому розділі детально розглядатимуться основні підходи, включаючи пояснення ключових понять, їхні переваги, недоліки та приклади використання.

API (інтерфейс прикладного програмування) — це набір протоколів, інструментів і правил, які дозволяють різним програмам взаємодіяти одна з одною. API надає доступ до сервісних функцій або даних без необхідності ручного втручання.

Як працює API?

API діє як посередник між користувачем і платформою, забезпечуючи взаємодію між системами. Наприклад, запит до автомобільного веб-сайту через API дозволяє отримати структуровану відповідь із необхідною інформацією, такою як список автомобілів. Запит зазвичай містить такі параметри, як тип автомобіля, ціновий діапазон або рік випуску, а відповідь складається з даних JSON або XML, які легко обробляються програмою.

API забезпечують швидкий доступ до великих обсягів структурованих даних, гнучкість у налаштуванні параметрів пошуку та легкість інтеграції завдяки детальній документації. Однак використання API може бути пов’язане з певними витратами, наприклад у вигляді платних тарифів або обмежень у безкоштовних

версіях. Існує також ризик блокування постачальника: зміни або відключення API можуть порушити інтеграцію.

Популярні приклади включають API Auto.ria, який дозволяє отримувати інформацію про автомобілі на основі різних параметрів, таких як марки, моделі та регіони. API Cars.com надає доступ до бази даних автомобілів у Сполучених Штатах.

Веб-скрейпінг

Веб-скрейпінг — це процес автоматичного збору даних із веб-сайтів. Програма імітує дії користувача, завантажуючи веб-сторінки, аналізуючи їх HTML-код і витягуючи потрібну інформацію. Цей підхід особливо корисний для отримання даних із платформ, які не надають API. Наприклад, збирати інформацію про ціни, характеристики продукту або відгуки клієнтів.

Для здійснення веб-скрейпінгу використовуються спеціальні інструменти. BeautifulSoup — популярна бібліотека для роботи зі структурою HTML, яка дозволяє ефективно витягувати елементи сторінки. Selenium забезпечує автоматизацію браузера, дозволяючи взаємодіяти навіть із динамічними веб-сторінками, створеними за допомогою JavaScript. Крім того, сучасні підходи включають використання Scrapy для створення масштабованих рішень та інтеграцію інструментів штучного інтелекту для покращення аналізу даних.

Веб-скрейпінг має свої переваги. Він дозволяє отримувати дані навіть із закритих платформ і адаптувати процес під конкретні завдання, наприклад, збирати інформацію лише про ціни чи популярні товари. Однак є і недоліки: часті зміни структури сторінок можуть викликати перебої в роботі скребка, а законодавчі обмеження на деяких сайтах можуть забороняти автоматичний збір даних з посиланням на умови використання.

Системи рекомендацій

Системи рекомендацій — це алгоритми, які аналізують уподобання користувачів і пропонують найбільш відповідні варіанти. Вони працюють різними підходами. Контент-орієнтовані системи аналізують характеристики

продукту, які цікавлять користувача, і пропонують подібні продукти. Спільна фільтрація використовує дані інших користувачів, ідентифікуючи популярні варіанти серед схожих груп. Гібридні системи поєднують ці підходи, досягаючи високої точності.

Такі алгоритми широко використовуються. Наприклад, Netflix аналізує історію переглядів, щоб рекомендувати схожі фільми, а автосервіси пропонують автомобілі, які відповідають запитам інших клієнтів. Системи рекомендацій покращують взаємодію з користувачем, персоналізуючи взаємодію з платформою та збільшуючи ймовірність покупки завдяки відповідним пропозиціям.

Однак ефективність рекомендацій залежить від кількості даних. Якщо інформації недостатньо, система може видати неточні результати. Крім того, складність алгоритмів і необхідність їх постійного вдосконалення потребують значних ресурсів.

Поєднання веб-скрейпінгу та систем рекомендацій відкриває великі можливості для бізнесу. Наприклад, скрепер може автоматично збирати дані про популярні продукти, а система рекомендацій аналізує цю інформацію та пропонує користувачам відповідні варіанти. Цей підхід використовується в електронній комерції, для аналізу ринку нерухомості, моніторингу цін конкурентів і навіть у сфері медицини, щоб рекомендувати ліки або процедури на основі історії.

Інтеграція цих технологій дозволяє платформам не тільки залишатися конкурентоспроможними, але й формувати інноваційні рішення для покращення взаємодії з користувачами.

Чат-боти та штучний інтелект(ШІ)

Чат-боти — це автоматизовані системи, які дозволяють користувачам взаємодіяти з ними за допомогою текстових або голосових повідомлень. Вони можуть приймати такі параметри пошуку, як «Знайти BMW до 30 000 доларів», і надавати відповіді в реальному часі з відповідними пропозиціями. Завдяки

інтеграції з такими платформами, як Dialogflow або Telegram Bot API, чат-боти забезпечують швидкий доступ до функцій безпосередньо в месенджерах.

ШІ та машинне навчання широко використовуються для вирішення складних завдань в автомобільній промисловості. Вони допомагають прогнозувати ціни, розпізнавати зображення для пошуку подібних варіантів і аналізувати ринкові тенденції. Такі інструменти, як TensorFlow і Scikit-learn, використовуються для створення та навчання моделей, які можуть ефективно обробляти великі обсяги даних і ідентифікувати корисні шаблони.

Наприклад, автомобільні платформи AI можуть аналізувати запити користувачів і автоматично вибирати найкращі варіанти автомобіля на основі заданих параметрів.

Розвиток таких технологій, як API, чат-боти та системи рекомендацій, значно покращує автоматизацію пошуку автомобілів. У кожного методу є свої сильні сторони: API надають доступ до даних, ШІ підвищує точність аналізу, а чат-боти полегшують взаємодію з користувачами. Поєднання цих інструментів дозволяє створювати інноваційні рішення, які відповідають запитам сучасного ринку.

Такі комплексні підходи не тільки допомагають підвищити ефективність пошуку, але й створюють персоналізований досвід для користувачів, що є ключовим фактором успіху в конкурентному середовищі.

Алгоритми класифікації та фільтрації

Алгоритми класифікації та фільтрації відіграють надзвичайно важливу роль у процесі автоматизації пошуку автомобіля. Це одна з ключових технологій, яка дозволяє надавати користувачам найбільш релевантну інформацію, усуваючи непотрібні або нерелевантні результати. Його основна ідея полягає в тому, щоб розділити великий обсяг даних на групи за певними критеріями, а також відфільтрувати дані, які не відповідають очікуванням користувача.

Ці алгоритми базуються на використанні сучасних досягнень у сфері штучного інтелекту та машинного навчання, хоча в основі їх лежать ідеї, досить прості та зрозумілі навіть людині, далекій від технологій. Велика кількість

оголошень про продаж автомобілів, які з’являються щодня: різні марки, моделі, роки випуску, ціни, пробіг, регіони продажу та багато інших параметрів. Для користувача, який шукає певний автомобіль, перегляд сотень і навіть тисяч таких оголошень стає надзвичайно складним завданням.

Коли користувач відкриває платформу пошуку автомобілів і вводить свої параметри, наприклад марку, модель, ціновий діапазон або пробіг, алгоритми починають аналізувати доступні списки. Вони класифікують дані, визначають, які автомобілі можуть відповідати його потребам, і відфільтровують інші, які не відповідають цим критеріям.

Однією з важливих характеристик цих алгоритмів є їх здатність «навчатися». Якщо, наприклад, користувачі часто ігнорують певну рекламу або, навпаки, зупиняються перед нею, система аналізує цю поведінку. Завдяки цьому з часом алгоритм починає краще розуміти, що люди шукають, і автоматично коригує свої рекомендації. Іншими словами, з кожним днем він стає «розумнішим».

Такі алгоритми не тільки допомагають знайти потрібний автомобіль, але й допомагають уникнути шахрайства. Система помічає, що в рекламі з’являється однаковий текст або схожі фотографії використовуються для різних автомобілів. Це може бути сигналом того, що щось не так. Алгоритм аналізує подібні випадки, порівнює їх з даними про шахрайську рекламу, яка була видалена в минулому, і може автоматично повідомляти адміністрацію платформи або навіть самостійно видаляти підозрілі записи.

Ще одна цікава особливість цих систем полягає в тому, що вони враховують індивідуальні переваги. Наприклад, якщо користувач раніше шукав автомобіль певної марки, система запам'ятає це і може запропонувати йому схожі варіанти, навіть якщо він не запитував.

Алгоритми класифікації та фільтрації працюють не лише з основними параметрами, такими як бренд чи ціна. Вони здатні аналізувати десятки, а часом і сотні дрібних деталей, які на перший погляд здаються неважливими.

Наприклад, це може бути колір автомобіля, його комплектація або навіть стиль фотографій, використаних в оголошенні.

Таким чином, алгоритми класифікації та фільтрації є основою сучасних платформ пошуку автомобілів. Вони роблять процес пошуку швидким, зручним і безпечним, усуваючи всі непотрібні кроки, які можуть перешкодити знаходженню ідеального результату. Завдяки їм навіть величезна кількість даних перестає бути перешкодою і стає цінним ресурсом, який допомагає приймати правильні рішення.

Обробка зображень та розпізнавання об’єктів

Обробка зображень і розпізнавання об'єктів є одним з найсучасніших і технологічно складних засобів, які активно використовуються для автоматизації пошуку автомобілів. Це галузь, заснована на використанні алгоритмів комп’ютерного зору, які дозволяють не тільки аналізувати зображення, але й отримувати з них структуровану інформацію. Основне завдання цієї технології — автоматична ідентифікація ключових характеристик автомобілів, представлених на фотографіях, і подальше їх використання для створення ефективних пошукових систем.

Ключовим елементом цієї технології є здатність алгоритмів розпізнавати об’єкти на зображеннях, класифікувати їх і аналізувати певні характеристики. Наприклад, вони можуть визначити марку, модель, колір автомобіля, а також визначити візуальні характеристики, такі як наявність пошкоджень, тюнінг або певний тип освітлення. Це досягається завдяки поєднанню методів машинного навчання та глибоких нейронних мереж, які дозволяють моделювати складні шаблони та залежності візуальних даних.

Технологія обробки зображень починається з етапу попередньої обробки, який включає усунення шумів зображення, корекцію кольору, нормалізацію розміру та виділення ключових областей. Ці процеси створюють основу для подальшого аналізу, дозволяючи алгоритму зосередитися на відповідних деталях, таких як форма автомобіля чи його номерний знак. У процесі аналізу

використовуються різні методи сегментації, які ділять зображення на окремі області, такі як кузов, фари, колеса тощо.

Наступний крок — розпізнавання предметів. Нейронні мережі, спеціально навчені на великих наборах зображень автомобіля, аналізують дані та співвідносять їх із раніше спостережуваними закономірностями. Для цього використовуються спеціальні архітектури, такі як згорточні нейронні мережі (CNN), які ідеально підходять для роботи із зображеннями. Наприклад, якщо алгоритм навчався на тисячах фотографій BMW X5 різних років, він зможе з високою точністю ідентифікувати цю машину на новому зображенні.

Розпізнавання об'єктів також допомагає визначити стан автомобіля. Наприклад, система може аналізувати наявність подряпин, вм'ятин або інших дефектів, порівнюючи текстуру зображення з базовими зразками. Тут використовуються методи аналізу текстури, зміни контрасту та виявлення аномалій. Це дозволяє автоматично оцінювати стан автомобіля, що особливо важливо для вторинного ринку, де точність опису часто залишає бажати кращого.

Крім того, алгоритми обробки зображень можуть бути інтегровані з геоінформаційними системами для визначення місцезнаходження автомобіля. Наприклад, аналіз фону фотографії може визначити тип місцевості або конкретні об’єкти, які можуть вказувати на місцезнаходження фотографії. Це може бути корисно для перевірки автентичності оголошення або пошуку викрадених автомобілів.

Іншим важливим аспектом є використання обробки зображень для покращення взаємодії з користувачем. За допомогою цієї технології можна створювати візуальні фільтри, які дозволяють користувачам шукати автомобілі за зовнішнім виглядом, а не лише за текстовими характеристиками. Наприклад, система може запропонувати варіанти, які нагадують завантажене зображення, використовуючи алгоритми отримання зображень на основі вмісту.

Високий рівень автоматизації, який пропонують ці технології, значно зменшує потребу в ручній обробці даних. Завдяки розпізнаванню об’єктів

користувач може аналізувати тисячі фотографій за лічені хвилини, роблячи процес швидким, ефективним і масштабованим. Це важливо в умовах, коли обсяг інформації в Інтернеті зростає з кожним днем.

Таким чином, обробка зображень і розпізнавання об'єктів є одними з фундаментальних компонентів автоматизованих систем пошуку автомобіля. Він дозволяє глибоко аналізувати візуальні дані, забезпечуючи точність, швидкість і новий рівень інтелектуальної взаємодії з інформацією. Ця технологія продовжує розвиватися, відкриваючи нові горизонти в оптимізації процесу пошуку та аналізу автомобіля.

1.2. Аналіз API сервісів для роботи з інформацією про продаж автомобілів та розгляд ключових факторів

API (інтерфейс прикладного програмування) є одним із найефективніших і універсальних інструментів для автоматизації пошуку автомобілів. Це технологія, яка дозволяє різним програмам взаємодіяти одна з одною, забезпечуючи обмін даними та функціями без необхідності доступу до внутрішньої структури чи деталей реалізації. Через API створюється міст між різними службами, що відкриває широкі можливості для інтеграції та автоматизації процесів[40-43].

Основною причиною, чому API настільки важливий для завдань автоматизації, є його здатність надавати доступ до великих обсягів даних безпосередньо з серверів, на яких вони зберігаються. У випадку з пошуком автомобіля це можуть бути бази даних рекламних майданчиків, офіційні сайти автовиробників або інші джерела, які публікують актуальну інформацію про автомобілі. Використання API дозволяє отримувати ці дані швидко, ефективно та в стандартизованому форматі.

Однією з найбільших переваг API є його масштабованість. API забезпечують прямий доступ до структурованих даних, які можна обробляти автоматично з мінімальним втручанням людини. Наприклад, API дозволяють отримувати інформацію про модель автомобіля, технічні характеристики, ціни, розташування дилерів, фотографії та навіть додаткові параметри, такі як пробіг або рік випуску.

Іншою важливою особливістю API є те, що вони забезпечують актуальність даних. Інформація, отримана через API, зазвичай надходить безпосередньо з серверів, які постійно оновлюються. Це означає, що користувачі автоматизованої системи завжди матимуть доступ до найновіших даних, не ризикуючи працювати із застарілою або неповною інформацією.

API також забезпечують велику гнучкість у налаштуванні пошуку. Вони дозволяють вказати детальні параметри, які слід враховувати під час запиту, наприклад ціновий діапазон, бажані марки чи моделі, регіон пошуку тощо. Це

зменшує кількість непотрібних даних і забезпечує точність результатів. Крім того, API підтримують автоматизацію повторюваних завдань, таких як регулярний моніторинг нових оголошень, створення аналітичних звітів або повідомлень про зміни в базі даних.

Однією з основних причин вибору API для автоматизації є його надійність. На відміну від інших методів, таких як веб-скрейпінг, API є офіційним способом доступу до даних, наданих платформою чи службою. Це означає, що дані, отримані через API, не будуть заблоковані та не призведуть до конфліктів із політикою використання служби. Крім того, платформи, які надають API, зазвичай зберігають документацію, яка спрощує інтеграцію та використання.

Безпека – ще один важливий аспект. API дозволяє контролювати доступ до даних за допомогою механізмів автентифікації та авторизації, таких як ключі API або маркери доступу. Це гарантує, що лише авторизовані системи можуть отримати доступ до інформації, що особливо важливо при роботі з конфіденційними даними або комерційними платформами.

API також є важливим інструментом для створення аналітичних систем. Його можна використовувати для збору даних для прогнозування, аналізу ринку або систем рекомендацій. Наприклад, інтеграція з API різних платформ дозволяє створити єдину базу даних, яка об’єднує інформацію з кількох джерел, що робить процес пошуку автомобіля більш повним та ефективним.

Таким чином, API є незамінним інструментом для автоматизації пошуку автомобілів завдяки своїй гнучкості, надійності, актуальності даних і масштабованості. Використання API не тільки значно прискорює процес пошуку, але й забезпечує високу точність результатів, що робить його оптимальним вибором для інтеграції в сучасні автоматизовані системи.

Auto.ria API — це інтерфейс для доступу до структурованих даних про автомобілі з платформи auto.ria.com. Він забезпечує інтеграцію функцій пошуку, фільтрації та відображення реклами в сторонні програми та сервіси. З його допомогою можна отримувати актуальні оголошення про продаж автомобілів, фільтрувати їх за такими параметрами, як марка, модель, рік випуску або тип

кузова, а також отримувати детальну інформацію про окремі автомобілі або довідкові дані, наприклад, списки марок або типів кузова.[9]

Офіційна документація API містить опис усіх доступних кінцевих точок, параметрів запиту та формат відповіді у форматі JSON, що забезпечує легкість використання для розробників.

Використання API Auto.ria має кілька переваг. Він забезпечує швидкий доступ до великої кількості актуальних даних, забезпечує гнучкість налаштування запитів і дозволяє працювати з оновленими оголошеннями. Однак варто враховувати, що безкоштовна версія API має обмеження на кількість запитів, і для роботи необхідно отримати ключ доступу.

Основні кінцеві точки:

- GET /categories/. Використовується для отримання списку категорій товарів. У контексті продажу автомобілів дозволяє визначити доступні категорії (автомобілі, вантажівки, легкові автомобілі тощо). Приклад відповіді[9]:

```
[
    {
        "category_id": 1,
        "name": "Легкові автомобілі"
    },
    {
        "category_id": 2,
        "name": "Мотоцикли"
    }
]
```

- GET /brands/. Отримання списку марок автомобілів. Параметри запиту: category_id — ідентифікатор категорії. Приклад відповіді:

```
[
    {
        "brand_id": 79,
        "name": "Toyota"
    },
    {
        "brand_id": 15,
        "name": "BMW"
```

```
    }
]
```

- GET /search/. Основна кінцева точка для пошуку оголошень. Параметри запиту:
  - marka_id — ідентифікатор марки;
  - model_id — ідентифікатор моделі;
  - price_ot і price_do — ціновий діапазон;
  - year_ot і year_do — діапазон років випуску.

  Приклад відповіді:

```
{
  "result": [
    {
      "id": 12345,
      "title": "Toyota Camry 2015",
      "price": 15000,
      "location": "Київ",
      "link":
"https://auto.ria.com/auto_toyota_camry_12345.html"
    },
    {
      "id": 67890,
      "title": "Toyota Corolla 2018",
      "price": 18000,
      "location": "Львів",
      "link":
"https://auto.ria.com/auto_toyota_corolla_67890.html"
    }
  ]
}
```

Конкуренти:

- Cars.com API (США):
  - Орієнтований на ринок США.
  - Забезпечує аналогічний функціонал, але має більше фільтрів для пошуку.
- Autotrader.co.uk API (Велика Британія):
  - Пропонує пошук за регіонами країни.

- Дозволяє інтегруватися з інструментами оцінки вартості авто.

1.1. Порівняльна таблиця

Характеристика	Auto.ria.com	Cars.com	Autotrader.com.uk
Географічне охоплення	Україна	США	Велика Британія
Формат відповідей	JSON	JSON	JSON
Наявність фільтрів	Так	Так	Так
Вартість	Безкоштовний тариф із обмеженнями	Платний	Платний

API auto.ria.com є ефективним інструментом для автоматизації пошуку автомобілів. Він забезпечує широку функціональність, релевантність даних і гнучкість налаштування параметрів. Незважаючи на деякі обмеження (зокрема, ліміти на запити), його використання доцільне для розробників, які створюють сервіси для роботи з інформацією про продаж автомобілів в Україні.

Ефективність автоматизації пошуку автомобіля багато в чому залежить від ряду технічних, організаційних і користувальницьких факторів. У цьому розділі розглянуто основні фактори, що впливають на якість і швидкість роботи автоматизованих систем у сфері обробки даних про продаж автомобілів.

Якість і повнота даних є важливими для належного функціонування автоматизованих систем. Дані, отримані через API або веб-скрейпінг, повинні мати чітку структуру, оскільки погана організація ускладнює їх обробку,

збільшуючи час виконання запитів. Актуальність також відіграє ключову роль: дані про автомобілі, їх ціни та технічний стан повинні регулярно оновлюватися. Застаріла інформація може ввести користувачів в оману та знизити довіру до системи. Надійність API залежить від його швидкості, мінімальної затримки та стабільної роботи навіть за високого навантаження.

Ефективність автоматизації тісно пов’язана з технічними ресурсами. Важливу роль відіграє продуктивність сервера: малопотужні системи можуть не впоратися з великим потоком запитів, що значно знижує швидкість і якість обробки. Якщо система працює у віддаленому середовищі, затримки мережі також можуть негативно вплинути на час відповіді. Важливим аспектом є масштабованість, оскільки платформа має здатність адаптуватися до збільшення трафіку, особливо під час пікових навантажень.

Технології, які використовуються для автоматизації, визначають її ефективність. API є оптимальним рішенням для роботи зі структурованими даними, оскільки мінімізують витрати на обробку інформації. Веб-скрейпінг використовується у випадках, коли API недоступні або їх функціональні можливості обмежені, але для цього потрібні швидкі та гнучкі парсери, здатні адаптуватися до змін у структурі веб-сторінок і уникати блокування сайтами.

Поєднання добре структурованих даних, високопродуктивних серверів і надійних технологій автоматизації дозволяє досягти високої ефективності системи навіть в умовах високого навантаження.

Кешування

Механізми кешування відіграють ключову роль у підвищенні ефективності автоматизації. Використання кешу може значно зменшити кількість повторних запитів до API або веб-ресурсів, прискоривши операції та зменшивши навантаження на систему. Це особливо важливо для платформ, які обробляють велику кількість запитів у режимі реального часу.

Безпека є невід’ємною частиною автоматизованих систем, особливо якщо вони працюють з особистою або діловою інформацією. Захист ключів API є

пріоритетом, оскільки їх викрадення може призвести до несанкціонованого доступу. Усі запити API мають бути захищені, а ключі доступу мають зберігатися в безпечному середовищі. Механізми автентифікації та авторизації повинні бути реалізовані для обмеження доступу до конфіденційних даних.

Ще одним важливим аспектом є шифрування даних. Інформація, що передається між клієнтом і сервером, повинна бути надійно зашифрована, щоб забезпечити конфіденційність і запобігти перехопленню.

Ефективна автоматизація базується на цілісному підході, який включає високоякісні дані, надійну інфраструктуру, передові технології, зручність використання та високий рівень безпеки. Врахування цих факторів дозволяє створити сучасну та продуктивну систему, яка відповідає вимогам користувача та сприяє зручному пошуку автомобіля.

РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ ТА АРХІТЕКТУРА ПІДСИСТЕМИ

2.1. Формалізація задачі автоматизації пошуку автомобілів.

Формалізація завдання з автоматизації пошуку автомобіля є необхідним етапом, який дозволяє скласти чітке уявлення про кінцеву мету та необхідний функціонал системи. Для забезпечення ефективного функціонування платформи необхідно враховувати специфіку завдання, очікувані результати та обмеження, які можуть виникнути під час його реалізації.

Автоматизація пошуку автомобіля призначена для вирішення ряду проблем, з якими стикаються користувачі під час виконання пошуку вручну. Наприклад, ручний пошук реклами на різних платформах займає багато часу, не завжди дозволяє знайти релевантні дані, а також ускладнюється різноманітністю фільтрів і критеріїв, які пропонують різні сайти. Тому автоматизована система повинна інтегрувати джерела інформації, мати інтуїтивно зрозумілий інтерфейс і оперативно реагувати на зміни в базах даних.

Загальні вимоги до автоматизації пошуку автомобіля формуються на основі аналізу потреб користувачів. Система повинна забезпечувати швидкий доступ до актуальної інформації про продажі автомобілів, дозволяючи налаштувати такі параметри пошуку, як марка, модель. Крім того, він має бути достатньо гнучким, щоб легко адаптуватися до оновлень сторонніх служб і потреб користувачів.

Для точного опису задачі можна використати таблицю, яка узагальнює основні характеристики:

2.1. Основні характеристики.

Елемент задачі	Опис
Мета системи	Автоматизувати процес пошуку автомобілів за заданими критеріями, зекономивши час користувачів.
Джерела даних	API популярних платформ (наприклад, auto.ria.com), веб-скрейпінг інших ресурсів.

Користувацький функціонал	Пошук за параметрами, сортування, отримання статистики ринку.
Обмеження	Ліміти на кількість запитів до API, необхідність підтримки оновлень структур веб-сайтів.

Одним із ключових компонентів формалізації проблеми є визначення входів і виходів системи. В якості вхідних даних система отримує запити від користувача у вигляді критеріїв пошуку (наприклад, «автомобіль 2020 року випуску, ціна до 15 000 $»). На виході користувач отримує структуровану інформацію: список актуальних оголошень з можливістю перегляду технічних характеристик, фотографій і контактів продавців.

Існує два основних підходи до збору даних: інтеграція з API-сервісами або веб-скрейпінг. API дозволяє отримувати дані швидко і в зручному форматі, але вимагає дотримання встановлених обмежень, наприклад обмеження на кількість запитів за одиницю часу. Веб-скрейпінг, у свою чергу, є альтернативою, але вимагає значно більшої обчислювальної потужності та може бути нестабільним через зміни в структурі веб-сайтів.

Важливим аспектом є організація отриманих даних. Всі автомобілі повинні бути представлені у вигляді єдиної конструкції, яка включає наступні основні параметри:

- Назва моделі;
- Опис
- Ціна;
- Додаткова інформація (фото, посилання на оголошення).

Серед факторів, що впливають на формалізацію завдання, варто виділити:

- Швидкість системи. Важливо забезпечити мінімальні затримки між запитом користувача та отриманням результатів.

- Релевантність даних. Система повинна періодично оновлювати інформацію, щоб уникнути ситуацій, коли реклама з’являється після її видалення.
- Юзабіліті. Інтерфейс повинен бути інтуїтивно зрозумілим навіть для користувачів без технічного досвіду.

Тому формалізація завдання дозволяє створити чітке уявлення про функціональні можливості майбутньої системи, її вимоги та обмеження. Це важлива передумова для розробки ефективного інструменту для автоматизації пошуку автомобіля.

2.2. Обґрунтування вибору методів і інструментів для реалізації задачі.

Розробка автоматизованого Telegram-бота для пошуку автомобілів вимагає ретельного вибору стеку технологій. У цьому пункті детально розглянуто основні інструменти, які використовуються в проєкті, пояснюючи їх призначення та переваги.

Java є об'єктно-орієнтованою мовою програмування, однією з найбільш використовуваних у світі. Було обрано для впровадження бота з наступних причин[4]:

- Портативність: програми Java працюють на будь-якій платформі, яка підтримує JVM;
- Масштабованість: Java підходить як для невеликих програм, так і для великих систем;
- Екосистема: Java має величезну кількість бібліотек і фреймворків, які спрощують розробку;
- Безпека: завдяки вбудованому управлінню пам’яттю та елементам керування виконанням Java є безпечним середовищем розробки.

Java[4, 21, 22] — це потужна та універсальна мова програмування, яка широко використовується для розробки різноманітних програмних рішень, від мобільних додатків до серверних і веб-систем. Однією з головних причин вибору Java для розробки цього проекту є її багатий набір функцій, висока

продуктивність, надійність і стабільність. Вона активно підтримує принципи об'єктно-орієнтованого програмування (ООП), що робить її ідеальним вибором для створення масштабованих програм, які легко підтримувати. Використання ООП у Java пропонує можливість модульної розробки, де кожен компонент програми (клас або об’єкт) має чітко визначену відповідальність. Це важливо для проєкту, оскільки він включає кілька ключових компонентів, які взаємодіють один з одним, але кожен із них виконує свою окрему роль.

Важливим аспектом є використання інкапсуляції: дані та методи, які з ними працюють, можуть бути приховані в класах і доступні лише через визначені інтерфейси. Це допомагає зменшити залежність між компонентами та забезпечує високий рівень безпеки та гнучкості. У проєкті кожен модуль (наприклад, обробка запитів API, збереження даних користувача, обробка помилок) має чітко визначену відповідальність і взаємодіє з іншими через чітко визначені інтерфейси.

Java також підтримує успадкування, що дозволяє створювати ієрархії класів і повторно використовувати код. Це зменшує кількість дублювання коду та забезпечує високу модульність. Завдяки цьому додавання нових функцій або зміна існуючих функцій можна виконувати без значних змін решти програми.

Іншою важливою перевагою Java є її незалежність від платформи, яка досягається за рахунок використання байт-коду та віртуальної машини Java (JVM). Програму, написану на Java, можна запускати на будь-якій платформі, яка підтримує JVM, без необхідності змінювати сам код. Це важливо для проєкту, оскільки бот повинен взаємодіяти з різними зовнішніми службами через API, і він повинен працювати на різних платформах, включаючи сервери та локальні машини.

Java також має дуже потужну систему обробки помилок, що особливо важливо для проектів, які вимагають надійності. Можливість обробки винятків, наявність потужних механізмів журналювання та налагодження дозволяють швидко виявляти та виправляти помилки, що робить програму більш стійкою до непередбачених ситуацій.

Ще однією великою перевагою Java є її масштабованість. Оскільки для проєкту потрібна ефективна обробка великих обсягів даних (наприклад, пошук автомобілів за різними критеріями), Java має потужні інструменти для роботи з потоками даних і асинхронними операціями. Це дозволяє скоротити час очікування користувача та підвищити ефективність програми. Крім того, Java підтримує багатопотоковість, що дозволяє виконувати кілька завдань одночасно, не блокуючи основний процес.

Java також підтримує безпечне програмування. Завдяки жорстким типам даних і суворим правилам компіляції Java зменшує кількість помилок, пов’язаних із неправильним використанням даних. Це дозволяє уникнути багатьох типових помилок програміста, які можуть призвести до непередбачуваних результатів або навіть зловмисних атак.

Таким чином, вибір Java для цього проєкту є абсолютно виправданим. Його об’єктно-орієнтований підхід, надійність, продуктивність і масштабованість роблять його ідеальним інструментом для розробки складних і високонавантажених програм, таких як бот Telegram для пошуку автомобілів. Завдяки величезній екосистемі бібліотек і можливості інтеграції з іншими службами Java забезпечує високу гнучкість і легкість розробки, дозволяючи зосередитися на реалізації основних функцій бота.

Spring Boot[15, 16, 17] є одним із найпопулярніших фреймворків для розробки Java, що забезпечує швидкий старт із готовими конфігураціями для типових завдань.

У цьому проєкті Spring Boot служить основою для серверної частини бота Telegram, забезпечуючи модульність і легку інтеграцію з іншими інструментами, такими як Spring Data JPA або Spring Web. Використання вбудованого сервера дозволяє запускати додаток без додаткових налаштувань, а наявність великої спільноти та якісної документації спрощує розробку та підтримку.

MySQL було обрано як реляційну базу даних для зберігання даних бота через її легкість інтеграції з Spring Boot, швидку роботу з великими обсягами

даних і підтримку складних запитів. Це популярна система з відкритим кодом, яка також пропонує гнучкість у масштабуванні.

Telegram Bots — це спеціалізовані програми, що працюють у месенджері Telegram, що дозволяють автоматизувати різноманітні завдання та інтерактивну взаємодію з користувачами. Їхня популярність зростає завдяки простоті використання, широкому функціоналу та можливості інтеграції з іншими сервісами. У контексті автоматизації пошуку автомобілів Telegram Bot може стати не тільки ефективним інструментом для роботи з даними, але й інтуїтивно зрозумілим інтерфейсом для кінцевих користувачів.

Бот працює безпосередньо в месенджері Telegram, який є одним із найпопулярніших серед користувачів у всьому світі. Це дозволяє швидко запустити систему без необхідності створення окремого сайту чи мобільного додатку. Користувачі можуть отримати доступ до бота через смартфон, планшет або комп’ютер, що робить систему максимально доступною.

Бот забезпечує інтерактивну взаємодію: користувач вводить запити, отримує відповіді в режимі реального часу, переглядає результати пошуку транспортних засобів у зручному форматі. Наприклад, бот може надсилати фотографії автомобілів, технічні характеристики, ціни і навіть контакти продавця, забезпечуючи зручність і швидкість пошуку.

Telegram Bot можна інтегрувати із зовнішніми службами, такими як автомобільні бази даних або API популярних рекламних платформ. Це означає, що користувачі можуть отримувати актуальну інформацію та постійно оновлюватись безпосередньо через бота. Завдяки цьому система може запропонувати пошук за різними параметрами: марка, модель, пробіг тощо.

Крім того, бот може виконувати функції «розумного посередника», автоматично обробляючи запити користувачів, форматуючи їх відповідно до формату API і повертаючи результати в зручному вигляді. Наприклад, якщо користувач хоче знайти автомобіль у певному регіоні за певний бюджет, бот перетворює це в API-запит і відображає відповідні результати.

Вони можуть бути оснащені алгоритмами штучного інтелекту, які аналізують запити користувачів і надають рекомендації. Наприклад, бот може вивчати поведінку користувачів, зберігаючи їхні попередні запити та пропонуючи варіанти, які найкраще відповідають їхнім інтересам. Це може бути особливо корисним для користувачів, які часто шукають транспортні засоби зі схожими характеристиками.

Бот також може забезпечити автоматичне сповіщення про нові оголошення. Наприклад, якщо в базі з’явиться автомобіль, який відповідає критеріям користувача, бот надішле сповіщення з усією необхідною інформацією. Це значно підвищує ефективність пошуку, особливо на динамічних ринках.

Telegram надає повний набір інструментів для створення ботів, включаючи API ботів, документацію та приклади використання. Це спрощує розробку навіть для невеликих команд. Telegram Bot може бути реалізований на різних мовах програмування, таких як Python, Java, Node.js тощо, що дозволяє вибрати найбільш зручний інструмент для розробника.

Ще однією перевагою є те, що Telegram забезпечує стабільну інфраструктуру для ботів, включаючи хостинг і обробку запитів. Це дозволяє зосередитися на системній логіці, а не на підтримці сервера чи усуненні технічних проблем.

Telegram-боти працюють у месенджері, який має високий рівень безпеки. Дані користувача, що передаються через бота, захищені шифруванням. Це особливо важливо, якщо система використовує особисті дані або фінансові операції, наприклад, для бронювання автомобіля або передоплати. Крім того, боти можуть впроваджувати додаткові рівні безпеки, такі як автентифікація користувачів за допомогою маркерів або паролів. Це дозволяє створити систему, яка відповідає вимогам конфіденційності та захисту даних.

Для роботи з API auto.ria.com використовуються бібліотеки OkHttp і Gson. OkHttp і Gson — це дві популярні бібліотеки в екосистемі Java, які забезпечують ефективну обробку веб-запитів і обробку відповідей JSON. Разом вони

створюють потужний набір інструментів для створення систем, які взаємодіють із веб-сервісами, такими як API для платформ пошуку автомобілів.

OkHttp — високооптимізована бібліотека для виконання HTTP-запитів. Її популярність пояснюється простотою використання, гнучкістю та високою продуктивністю. Вона полегшує надсилання запитів на сервери API та отримання відповідей, підтримуючи синхронні та асинхронні операції. Це особливо важливо для сценаріїв, де потрібна швидка взаємодія з API.

На додаток до основних функцій, таких як запити GET і POST, OkHttp підтримує: налаштування заголовків запитів для автентифікації або передачі додаткової інформації, робота з HTTPS, включаючи сертифікати безпеки, обробка запитів з великими обсягами даних за допомогою потоків, кешування для оптимізації роботи з часто використовуваними даними.

Наприклад, у рамках автоматизації пошуку автомобілів OkHttp можна використовувати для надсилання запитів до API з такими параметрами, як марка автомобіля, модель або бюджет.

Gson — це бібліотека від Google, яка полегшує роботу з даними JSON. Він автоматично перетворює об’єкти JSON на класи Java і навпаки, спрощуючи обробку даних, які повертаються API.

JSON є основним форматом для передачі даних між веб-сервісами, оскільки його легко читати людям і машинам. Gson дозволяє розробникам уникнути виснажливого ручного аналізу JSON, замінивши його автоматичною десеріалізацією.

OkHttp і Gson ідеально доповнюють один одного. У типовому сценарії OkHttp використовується для виконання запиту, а Gson використовується для обробки отриманої відповіді. Такий підхід робить взаємодію з API швидкою та ефективною.

Це бібліотеки, які не потребують значних системних ресурсів. Вони добре задокументовані та підтримуються великою спільнотою розробників, що полегшує їх впровадження та підтримку. Таким чином, це найкращий вибір для розробників, які хочуть створювати сучасні, швидкі та надійні системи, такі як

платформи для автоматизованого пошуку автомобілів. Завдяки OkHttp і Gson розробка стає інтуїтивно зрозумілим процесом, який дозволяє зосередитися на бізнес-логіці, а не на технічних деталях.

Поєднання цих технологій створює ефективну, гнучку та масштабовану систему. MySQL забезпечує стабільне зберігання даних, Spring Boot підтримує надійність бекенда, а інтеграція з Telegram API дозволяє створити зручний інтерфейс користувача. Використання OkHttp і Gson забезпечує плавну інтеграцію з іншими зовнішніми службами для отримання та обробки необхідної інформації.

2.3. Постановка технічних вимог до розробленої системи та визначення критеріїв оцінки якості

Для ефективного впровадження системи автоматизації пошуку автомобіля за допомогою Telegram-бота необхідно чітко визначити технічні вимоги. Вони поділяються на функціональні та нефункціональні вимоги, які охоплюють аспекти функціонування системи, її продуктивність, безпеку та масштабованість.

Серед функціональних вимог ключовим є забезпечення інтерактивності бота. Бот повинен коректно обробляти команди користувача, забезпечуючи можливість пошуку автомобілів за різними параметрами, такими як марка, модель. Користувачам повинна бути доступна детальна інформація про обраний автомобіль із зазначенням його основних характеристик. Інтеграція з API рекламної платформи auto.ria.com забезпечить доступ до актуальних даних про автомобілі, а можливість підключення до кількох API розширить базу пошуку. Всі дані, отримані від користувачів, повинні зберігатися в тимчасових змінних для швидкої обробки, а логування запитів допоможе в подальшому аналізі та оптимізації роботи бота.

Нефункціональні вимоги спрямовані на забезпечення продуктивності, безпеки та масштабованості системи. Відповідь бота на запити користувача не повинна перевищувати двох секунд, навіть при обробці до сотні одночасних

запитів. Оновлення даних із зовнішніх API має відбуватися кожні п’ять хвилин, що гарантує їх актуальність. Система має бути доступною безперервно, 24 години на добу, з мінімальним часом простою, що не перевищує однієї години на місяць. Для забезпечення безпеки персональних даних користувачів усі дані, що передаються між ботом і сервером, повинні бути зашифровані. Захист ключів API від несанкціонованого доступу також є критично важливим аспектом.

Система має бути легко масштабованою, що дозволить збільшити кількість користувачів і розширити базу даних без істотних змін коду. Інтеграція нових функцій, таких як додаткові параметри пошуку або підключення нових платформ, має бути максимально простою. Інтерфейс бота повинен бути інтуїтивно зрозумілим для будь-якого користувача. Відповіді повинні бути надані в чітко структурованому форматі, включаючи назву автомобіля, ціну, основні характеристики, такі як рік випуску, пробіг і тип двигуна, а також посилання на оголошення. Це дозволить створити зручний та ефективний інструмент для пошуку автомобілів, які відповідають сучасним вимогам та очікуванням користувачів.

2.2. Схематичне представлення компонентів системи.

Компонент	Опис
Telegram-бот	Отримує запити від користувачів і повертає результати.
Spring Boot API	Серверна частина для обробки запитів і взаємодії з API.
MySQL база даних	Зберігає результати пошуку та історію запитів.

Обрані технічні вимоги забезпечують стабільність, гнучкість і функціональність розробленої системи. Дотримання цих вимог гарантує його ефективність у вирішенні проблеми пошуку авто через Telegram-бот.

Оцінка якості системи автоматизації пошуку автомобіля є важливим етапом, який дозволяє перевірити її відповідність вимогам і очікуванням

користувачів. Основними критеріями оцінки є функціональні, технічні та користувальницькі аспекти, які забезпечують комплексний аналіз системи.

Функціональні критерії включають вимоги до точності результатів, де відсоток правильних відповідей на пошукові запити має бути не менше 95%, а також необхідність відображення актуальних даних, таких як ціни та фотографії автомобілів. Повнота пошуку передбачає, що кількість знайдених результатів має відповідати першій сторінці пошукових систем сервісу, а в разі відсутності даних система повинна сповістити користувача.

Технічні критерії охоплюють швидкість роботи, зокрема час виконання запиту до API, який не повинен перевищувати 2 секунди. Також важливо, щоб система не відчувала значних затримок навіть при великій кількості одночасних запитів. Стабільність системи повинна забезпечувати відсутність помилок в стандартних умовах виконання запитів і можливість відновлення роботи після несподіваних помилок, таких як недоступність API. Масштабованість передбачає здатність системи обробляти зростаючий обсяг даних і кількість користувачів без значного зниження продуктивності, а також здатність інтегрувати нові служби API без зміни коду ядра.

Політика використання наголошує на простоті використання, де всі елементи керування мають бути інтуїтивно зрозумілими для користувачів бота Telegram. Результати пошуку мають відображатися в зручному вигляді з чіткою структурою, що містить такі дані, як назва автомобіля, ціна, опис і посилання на оголошення. Також важливо забезпечити високий рівень задоволеності користувачів, який можна оцінити позитивними відгуками та високою ймовірністю повторного використання системи. Система має бути доступною 24/7 з мінімальними простоями.

Безпека та конфіденційність включають захист даних, забезпечення конфіденційності пошукових запитів користувачів, а також захист переданих і збережених даних за допомогою шифрування. Система має бути захищена від поширених атак, таких як SQL-ін’єкції, що забезпечує її стійкість до шкідливих впливів.

2.3. Метрики для оцінки якості

Критерій	Метрика	Цільове значення
Точність результатів	Відсоток коректних відповідей	>= 95%
Швидкість обробки запитів	Середній час відповіді на запит	=< 2 секунди
Стабільність роботи	Частота збоїв під час виконання запитів	< 1 на 1000 запитів
Зручність використання	Відсоток позитивних відгуків користувачів	>= 90%
Масштабованість	Здатність обробляти додаткові навантаження	До 1000 запитів/год
Безпека даних	Кількість успішних атак	0

Встановлення чітких критеріїв оцінки якості роботи системи забезпечує відповідність розробленого Telegram-бота сучасним вимогам автоматизації. Це також допомагає покращити функціональність і задовольнити потреби кінцевих користувачів.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Архітектура та алгоритми роботи телеграм-бота для пошуку автомобілів.

Архітектура бота побудована на Java з використанням бібліотеки для взаємодії з Telegram API, що дозволяє ефективно та зручно реалізовувати функціонал для користувачів. Важливою особливістю цієї архітектури є її модульність і гнучкість, що забезпечує зручність розширення і підтримки системи в майбутньому. Ключовими компонентами архітектури є класи, моделі даних і механізми взаємодії, які забезпечують належне функціонування бота.

Одним з основних елементів системи є клас CarParametersSearch, який відповідає за обробку API-запитів і взаємодію із зовнішнім сервісом Auto.RIA. Цей клас забезпечує прямий доступ до таких функцій, як отримання списку марок автомобілів, моделей конкретної марки, а також дозволяє отримати детальну інформацію про автомобілі, які відповідають заданим параметрам пошуку. Використовуючи цей клас, бот може швидко надавати користувачам точну та актуальну інформацію про різні автомобілі на основі їхніх уподобань.

Дані, отримані через API, перетворюються в прості у використанні об'єкти, що дозволяє ефективно працювати з інформацією на всіх етапах взаємодії з користувачем. CarBrand — це шаблон, який містить основні характеристики марок автомобілів, такі як назва та код. Модель CarModel відповідає за зберігання інформації про конкретні моделі автомобілів, зокрема їх назви та коду, що дає можливість точно визначити, що це за автомобіль. Car — це більш повний шаблон, який містить всю необхідну інформацію про автомобіль, включаючи його опис, фотографії, посилання, ціну та місцезнаходження. Це дозволяє користувачеві отримати повну картину кожного об’єкта, який він шукає.

Клас користувача(User) є важливою складовою системи, оскільки містить інформацію про конкретного користувача Telegram, його поточний статус і обрані параметри пошуку. Використовуючи цей клас, бот зберігає контекст між діалогами, що дозволяє персоналізувати взаємодію. Це забезпечує зручність для

користувачів, оскільки бот може запам’ятовувати їхні вподобання та надавати більш відповідні рекомендації під час наступних сеансів взаємодії.

Іншим важливим елементом є список транспортних категорій (TransportCategory), який реалізується шляхом перерахування. Ця частина системи дозволяє користувачам вибрати тип пошуку, наприклад, автомобілі, мотоцикли, спецтехніка та інші категорії транспорту. Це дозволяє боту гнучко підлаштовуватися під різні запити користувачів, забезпечуючи зручність і ефективність роботи з різними типами транспортних засобів.

Загалом ця архітектура забезпечує високий рівень гнучкості, масштабованості та стабільності системи. Він дозволяє боту швидко адаптуватися до нових вимог і змін зовнішніх сервісів, роблячи систему довговічною і надійною.

Алгоритм бота створений для забезпечення ефективної та зручної взаємодії з користувачем, при цьому кожен етап роботи ретельно продуманий для плавного переходу між етапами. Після запуску бота користувач вводить команду /start, яка запускає процес взаємодії. Бот негайно створює або оновлює запис про користувача в базі даних, встановлюючи початковий статус користувача як FREE, тобто бот готовий почати діалог.

Першим важливим кроком у взаємодії є обробка запиту категорії транспорту. Коли користувач хоче виконати пошук, бот пропонує вибрати категорію транспорту, надаючи список доступних варіантів, таких як автомобілі, мотоцикли, спецтехніка та інші. Коли користувач вибирає категорію, статус його профілю оновлюється до WAIT_BRAND, а тимчасове значення категорії зберігається в базі даних, що дозволяє переходити до наступного кроку.

Після вибору категорії наступним кроком є вибір бренду. За допомогою методу fetchCarBrands бот отримує список брендів для обраної категорії та пропонує користувачеві вибрати потрібний варіант. Коли користувач вибирає бренд, статус змінюється на WAIT_MODEL і зберігається тимчасовий ідентифікатор вибраного бренду. Це дозволяє боту зберігати важливу інформацію та коректно продовжувати діалог.

Наступний крок - вибір моделі автомобіля. Коли користувач вибирає марку, бот завантажує список моделей цієї марки за допомогою методу fetchCarModel. Користувач вибирає шаблон, і в цей момент статус змінюється на FREE, що означає, що користувач завершив вибір і готовий отримувати результати пошуку.

Після вибору всіх параметрів бот переходить до пошуку автомобіля. За допомогою методу getCarIds бот шукає автомобілі, які відповідають обраним критеріям, після чого за допомогою getCarsByIds отримує детальну інформацію про знайдені автомобілі. Ці дані включають фото автомобіля, опис, ціну, місто розташування та посилання на сайт Auto.RIA для більш детальної інформації. Результати пошуку надсилаються користувачеві у вигляді повідомлень, що містять всю необхідну інформацію, організовану в зручному для перегляду форматі.

Така архітектура алгоритму дозволяє боту забезпечувати послідовний та інтуїтивно зрозумілий діалог з користувачем. Кожен етап взаємодії поступово уточнює параметри пошуку, надаючи користувачеві можливість вибору з кількох варіантів і отримання найбільш релевантних результатів. Завдяки цьому користувач може швидко і легко знайти автомобіль, який відповідає його потребам.

Телеграм-бот для пошуку автомобілів призначений для легкого та швидкого пошуку транспортних засобів, які відповідають конкретним запитам користувача. Основні функції бота включають вибір категорії транспортного засобу, де користувач має можливість вибирати з безлічі варіантів, таких як автомобілі, мотоцикли, водні транспортні засоби, спецтехніка, причепи, вантажівки, автобуси, кемпери, авіатранспорт і сільськогосподарські. техніка. Цей список категорій завантажується динамічно через API сервісу Auto.RIA, що допомагає забезпечити актуальність і повноту інформації для користувачів.

Наступним кроком буде вибір марки та моделі автомобіля. Вибравши категорію, користувач може ввести бренд. Бот перевіряє, чи існує цей бренд на платформі Auto.RIA, і якщо такий бренд знайдено, його ID тимчасово

зберігається в моделі користувача для подальшого використання. Потім користувач вводить модель вибраного бренду, і бот запускає аналогічний алгоритм пошуку, щоб знайти відповідні моделі на платформі

Після вибору моделі бот переходить до пошуку автомобілів, які відповідають заданим критеріям. Користувач отримує результати на основі обраної категорії автомобіля, марки та моделі. Це дозволяє точно і швидко знаходити автомобілі, що відповідають вимогам користувача.

Коли результати пошуку готові, бот надсилає користувачеві детальну інформацію про знайдені автомобілі в зручному форматі. До кожного авто додається фото, короткий опис, ціна (вказана в гривнях), місто, де знаходиться авто, а також посилання на повну інформацію на сайті RIA для більш детального перегляду.

При необхідності користувач може перезапустити пошук за допомогою відповідної кнопки. Якщо бот не знаходить автомобілі, що відповідають заданим параметрам, він надає користувачеві повідомлення про відсутність результатів. У цьому випадку користувачеві пропонується змінити критерії пошуку для отримання більш точних результатів.

Цей процес створює зручну та ефективну взаємодію між користувачем і ботом, дозволяючи швидко знайти потрібний автомобіль і при необхідності налаштувати параметри пошуку.

Основні модулі програми

Модульна структура Телеграм-бота для пошуку автомобілів передбачає кілька важливих компонентів, кожен з яких виконує певні функції для забезпечення ефективної роботи системи. Одним з основних модулів є модуль обробки повідомлень Telegram, який відповідає за взаємодію з користувачем через API Telegram. Основним класом цього модуля є CarSearchBot, який дозволяє записувати такі команди, як /start, обробляти вибір категорій, марок і моделей, а також надсилати результати пошуку користувачеві. Завдяки цьому

модулю бот може коректно відповідати на запити користувачів і надавати їм необхідну інформацію.

Ще один важливий компонент — модуль взаємодії з API Auto.RIA. Цей модуль дозволяє роботам отримувати дані автомобіля за допомогою зовнішнього API. Основним класом цього модуля є CarParametersSearch, який містить кілька основних методів. Метод fetchCarBrands отримує список брендів, fetchCarModel — список моделей даного бренду, getCarIds шукає автомобілі за заданими параметрами, а метод getCarsByIds дозволяє отримати детальну інформацію про знайдені автомобілі. Завдяки цьому модулю бот може здійснювати точний пошук автомобілів, відповідно до запитів користувачів.

Модуль зберігання даних користувача відіграє важливу роль у збереженні контексту між повідомленнями. Він відповідає за збереження обраної категорії, марки та моделі, а також іншої важливої інформації. Основним класом цього модуля є UserService, який гарантує, що ці дані зберігаються в базі даних для швидкого доступу. Це дозволяє боту відновлювати контекст і забезпечувати зручну роботу користувача.

Модуль моделі даних відповідає за структурування даних, отриманих від API, і їх внутрішнє представлення. Цей модуль містить кілька класів: CarBrand для інформації про бренд, CarModel для моделі автомобіля та Car для детальної інформації про автомобіль. Кожен клас відповідає за зберігання певної інформації, що дозволяє більш ефективно працювати з даними.

Модуль обробки помилок призначений для обробки помилок під час запитів до API або Telegram. У разі будь-якої проблеми, наприклад, недоступності API або некоректного формату введених даних, модуль інформує користувача про проблему. Це дозволяє зручніше та безпечніше користуватися ботом, зменшуючи ризик помилок і непорозумінь.

Усі ці модулі працюють разом, забезпечуючи велику гнучкість програми та полегшуючи її підтримку та розвиток. Така модульна структура дозволяє ефективно взаємодіяти з користувачами, здійснювати пошук автомобілів за

заданими параметрами та зберігати важливу інформацію для подальшого використання.

3.2. Взаємодія з API auto.ria.com: особливості реалізації.

Взаємодія з API auto.ria.com базується на REST-запитах до серверів платформи, які повертають дані у форматі JSON. Для отримання доступу використовується API-ключ, що передається як параметр у кожному запиті. Основними аспектами роботи з API є виконання пошукових запитів, отримання списків брендів, моделей, та аналіз отриманих оголошень.

Для реалізації цього функціоналу було використано бібліотеку RestTemplate, що дозволяє зручно працювати з HTTP-запитами у Java.

Основні методи для роботи з API Auto.RIA:

- Пошук оголошень(searchCars)

  Метод здійснює пошук автомобілів за заданими параметрами. Він використовує endpoint /search.

```
public String searchCars(String categoryId, String
brandId, String modelId) {
    String url = String.format(

"https://developers.ria.com/auto/search?api_key=%s
&category_id=%s&marka_id=%s&model_id=%s",
            API_KEY, categoryId, brandId, modelId
    );

    return restTemplate.getForObject(url,
String.class);
}
```

  - Вхідні параметри:
    - categoryId – ID категорії транспорту;
    - brandId – ID бренду автомобіля (наприклад, BMW, Audi);
    - modelId – ID моделі автомобіля.
  - Результат: JSON-рядок із результатами пошуку.

- Отримання списку моделей (getModels)

Метод отримує список моделей автомобіля для конкретного бренду через endpoint /categories/{category_id}/marks/{mark_id}/models.

```
public String getModels(int categoryId, String
brandId) {
    String url = String.format(

"https://developers.ria.com/auto/categories/%d/mar
ks/%s/models?api_key=%s",
            categoryId, brandId, API_KEY
    );

    return restTemplate.getForObject(url,
String.class);
}
```

- Вхідні параметри:
  - ❖ categoryId – ID категорії;
  - ❖ brandId – ID бренду.
- Результат: JSON-рядок із переліком моделей бренду.

Для інтеграції цих методів у бізнес-логіку програми, отримані JSON-результати можуть бути конвертовані у Java-об'єкти за допомогою бібліотеки Jackson:

```
ObjectMapper objectMapper = new ObjectMapper();
List<Model> models = objectMapper.readValue(getModels(1,
"12"), new TypeReference<>() {});
```

Цей підхід дозволяє динамічно обробляти відповіді від API та використовувати їх у функціоналі Telegram-бота. Наприклад, за допомогою методу getCarDetails можна надати користувачу детальну інформацію про автомобіль за запитом.

Методи класу CarParametersSearch покривають всі ключові аспекти взаємодії з API auto.ria.com: від пошуку автомобілів до отримання списків брендів, моделей та деталей оголошень. Вони оптимізовані для зручної інтеграції у функціонал Telegram-бота та забезпечують ефективний доступ до даних.

3.3. Тестування системи та аналіз ефективності

Тестування в програмуванні є ключовим кроком, який гарантує якість і надійність програмного забезпечення. Воно діє як своєрідний «захисний бар'єр», який допомагає запобігти багатьом проблемам ще до того, як програма досягне кінцевого користувача. Думка про те, що тестування — це лише перевірка на наявність помилок, надто спрощена. Його завдання набагато ширше: перевірити відповідність створеного продукту вимогам, його стабільність, правильність роботи в різних умовах, а також готовність до використання в реальних умовах.

Розробка програмного забезпечення без належного тестування може призвести до серйозних наслідків. Помилки в коді можуть спричинити несправності, втрату даних або навіть загрожувати безпеці користувача. Наприклад, у системах, що використовуються для автоматизації пошуку автомобілів, неправильна робота алгоритмів фільтрації або обробки запитів може зробити продукт непридатним для користувачів, які очікують точних і швидких результатів. Крім того, відсутність тестування може завдати шкоди репутації компанії, особливо якщо проблеми виникають після випуску продукту на ринок.

Тестування допомагає виявити помилки на ранній стадії розробки. Це надзвичайно важливо, оскільки виправлення помилок у готовому продукті зазвичай набагато дорожче і займає набагато більше часу. Крім того, зволікання з виявленням проблеми може призвести до того, що помилки почнуть «нагромаджуватися», що ускладнить підтримку та подальший розвиток системи. Ось чому тестування часто інтегрується безпосередньо в процес розробки, використовуючи методи автоматизованого тестування, які перевіряють код із кожною новою зміною.

Ще однією важливою перевагою тестування є перевірка того, як система працює за різних умов. Розробники не завжди можуть передбачити, як користувачі будуть взаємодіяти з програмою, які дані будуть вводити або в яких сценаріях працюватимуть. Тестування допомагає оцінити стійкість системи до

таких непередбачуваних ситуацій. Наприклад, якщо програма працює з API, тестування може виявити, як вона поводиться за повільного з'єднання, відсутності доступу до серверів або отримання неправильних відповідей.

Тестування також забезпечує гармонійну роботу різних компонентів програми. У складних системах, які використовують алгоритми машинного навчання, інтегрують зовнішні API та обробляють великі обсяги даних, важливо перевірити, чи всі ці елементи правильно взаємодіють один з одним. Наприклад, створюючи пошукову платформу автомобіля, потрібно переконатися, що модуль обробки зображень коректно взаємодіє з базою даних і що інтерфейс користувача правильно відображає результати пошуку.

Тестування є невід'ємною частиною розробки будь-якої програмної системи, особливо Telegram-ботів, які працюють з реальними користувачами. Мета тестів — перевірити, чи коректно працюють функціональні модулі, як бот реагує на різні сценарії використання, а також оцінити стабільність роботи під навантаженням.

Під час тестування автопошукового бота Telegram перевірялись його основні можливості: взаємодія з користувачем, обробка запитів до API auto.ria.com, перевірка некоректності даних і надання зручного інтерфейсу для пошуку оголошень. Тести проводилися з використанням реальних сценаріїв взаємодії, а результати були задокументовані за допомогою скріншотів.

Загальний підхід до тестування:

- Початок роботи: Перевірка стартової команди /start, яка ініціалізує користувача та показує доступний функціонал;
- Основний функціонал: Перевірка процесу пошуку оголошень, починаючи від вибору типу транспорту до отримання результатів;
- Обробка помилок: Тестування сценаріїв, де користувач вводить некоректні дані, наприклад, неправильні марку чи модель авто;
- Результати пошуку: Перевірка правильності виведення списку оголошень, включаючи зображення, опис, ціну та посилання;

- Стабільність: Оцінка роботи бота при багаторазових запитах і некоректних даних.

Результати тестування:

- Сценарій 1 (рис. 3.1): Початок роботи бота. Користувач ввів команду /start. У відповідь бот привітав його текстом:
  «Вітаю! Ви успішно зареєструвались в боті. Щоб почати користування, натисніть кнопку "Знайти оголошення".»

  Одночасно з цим бот надіслав інтерактивну клавіатуру з кнопкою "Знайти оголошення", яка стала основним способом взаємодії з ботом.

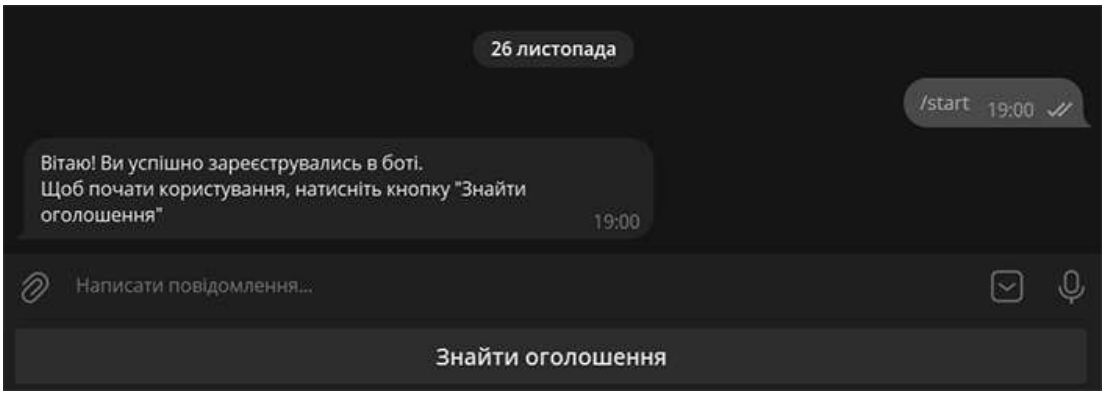


Рис. 3.1 – Screenshot сценарію 1

- Сценарій 2 (рис. 3.2): Вибір типу транспорту. Користувач натиснув кнопку "Знайти оголошення", після чого бот запитав:
  «Оберіть тип транспорту:»
  Було надано клавіатуру з усіма доступними типами транспорту.

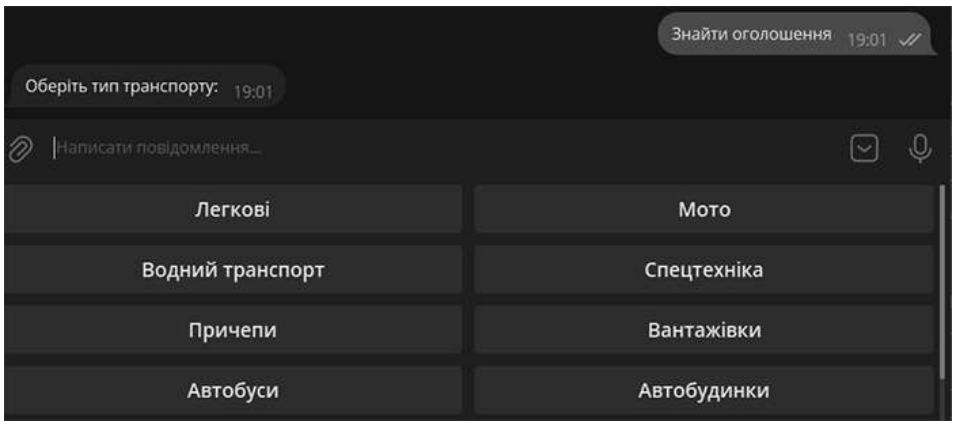


Рис. 3.2 – Screenshot сценарію 2

- Сценарій 3 (рис. 3.3): Введення некоректної марки авто. Користувач навмисно ввів некоректну марку: "BME". Бот проаналізував запит через API і повернув повідомлення:
  «Такої марки авто не знайдено, спробуйте ще раз:»

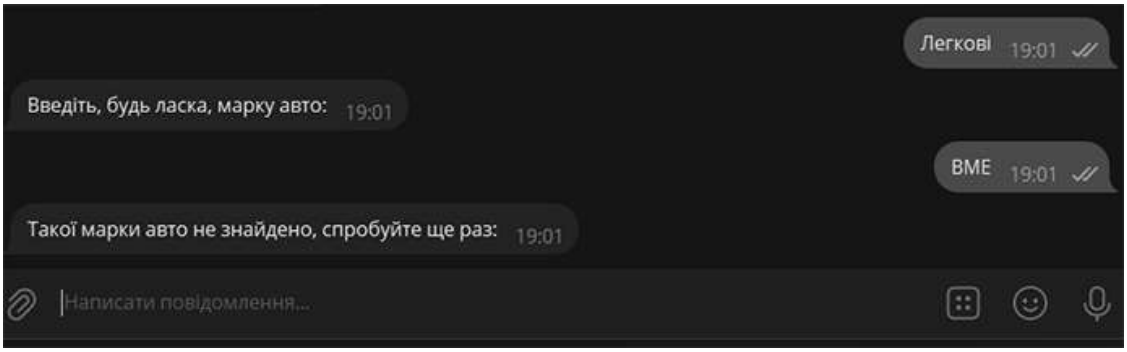


Рис. 3.3 – Screenshot сценарію 3

- Сценарій 4 (рис. 3.4): Введення некоректної моделі авто. Користувач ввів некоректну модель: "M%". Бот обробив це й надав повідомлення:
  «Такої моделі не знайдено, спробуйте іншу:»

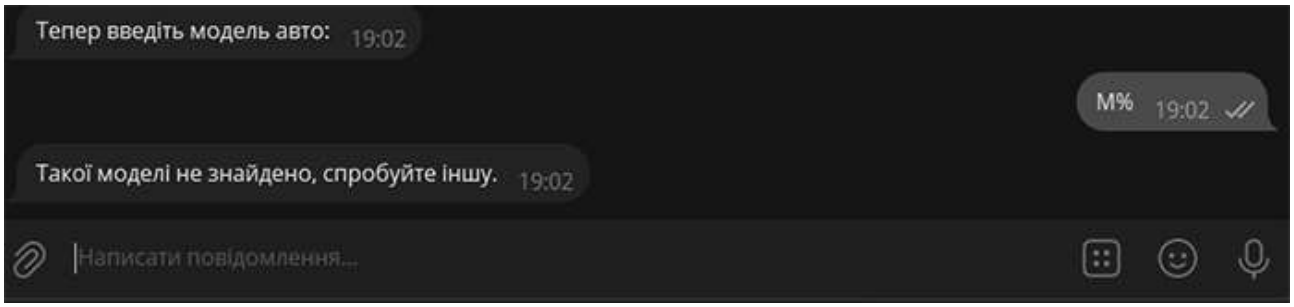


Рис. 3.4 – Screenshot сценарію 4

- Сценарій 5 (рис. 3.5-3.6): Отримання результатів пошуку. Бот повернув результати пошуку, кожен із яких містив:
  - Фото авто;
  - Назву оголошення;
  - Ціну;
  - Розташування;
  - Короткий опис.

- Посилання на повну версію оголошення. Останнє повідомлення в списку було фінальним оголошенням із відповідними деталями.

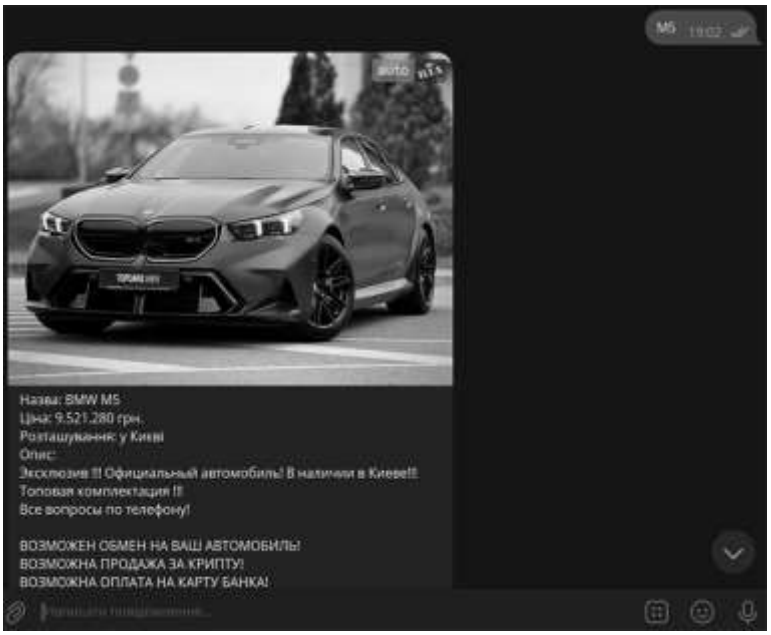


Рис. 3.5 – Screenshot сценарію 5(1 частина)

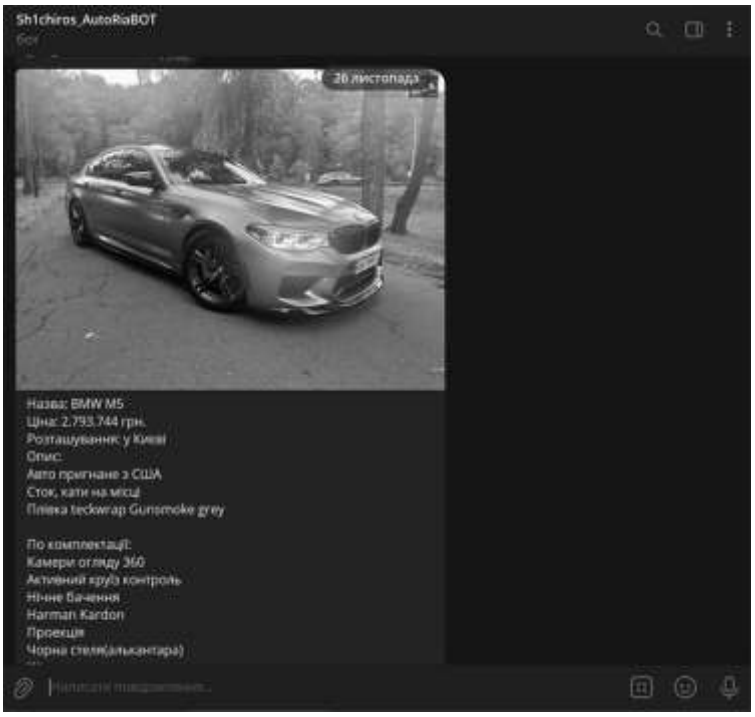


Рис. 3.6 – Screenshot сценарію 5(1 частина)

Результати тестування показали, що бот коректно виконує свої функції відповідно до очікувань. Важливо відзначити, що:

- Бот коректно реагує на неправильні введення, що покращує взаємодію з користувачем;

- Відображення результатів пошуку є інформативним і зручним для взаємодії;
- Всі API запити виконуються швидко, а відповіді формуються в зрозумілому форматі.

Надані скріншоти підтверджують працездатність бота в реальних умовах.

Аналіз ефективності розробленої системи є важливим етапом у процесі оцінки її ефективності та корисності в реальному середовищі. У цьому підрозділі розглядаються як технічні, так і практичні аспекти використання Telegram-бота, розробленого для пошуку автомобілів. Оцінка ефективності базується на таких параметрах, як швидкість роботи, зручність використання, точність результатів пошуку, а також здатність інтегруватися з іншими системами і забезпечення стабільної роботи при великих навантаженнях.

1) Швидкість роботи

Одним із ключових параметрів ефективності будь-якої програмної системи є її швидкість. Розроблений бот робить запити до API сайту auto.ria.com для отримання оновленої інформації про оголошення. Завдяки оптимізованому пошуку та обробці запитів бот забезпечує швидкий доступ до даних навіть при великій кількості запитів. Зверніть увагу, що час відповіді на запит користувача становить приблизно 2-4 секунди, що є оптимальним для взаємодії з користувачем. Це досягається завдяки таким факторам:

- Використання ефективного API для пошуку автомобілів;
- Зведіть до мінімуму непотрібні запити до серверів;
- Оптимізація обробки та фільтрація результатів.

2) Зручність використання

Однією з головних цілей розробки бота було створення зручного інтерфейсу для користувачів. Бот використовує клавіатуру для зручного введення параметрів пошуку, що забезпечує інтуїтивно зрозумілу взаємодію. Користувачі можуть вибрати вид транспорту, марку та модель автомобіля, а також переглянути оголошення, які відповідають заданим

критеріям. Відсутність необхідності вводити складні команди або знати точні параметри робить бота доступним для широкої аудиторії користувачів.

3) Точність результатів пошуку

Оскільки бот шукає рекламу через API auto.ria.com, точність результатів багато в чому залежить від актуальності та точності даних, що надходять із цього сервісу. Під час тестування бот продемонстрував високу точність результатів пошуку. Користувачі отримують список автомобілів, які точно відповідають заданим параметрам, таким як марка, модель, ціна, місцезнаходження тощо. Однак деякі проблеми можуть виникнути в ситуаціях, коли API не може знайти рекламу за певним запитом, що, однак, є рідкісним випадком.

4) Налаштування та розширюваність

Розроблений бот дуже розширюваний і може бути адаптований до інших категорій товарів чи послуг. За допомогою простих налаштувань можливо змінювати джерела інформації, додавати нові категорії або змінювати формат відображення результатів. Така гнучкість дозволяє використовувати бота не тільки для пошуку автомобілів, а й для пошуку інших товарів чи послуг.

5) Стійкість і навантаження

Щоб забезпечити стабільну роботу під великим навантаженням, бот повинен мати можливість обробляти декілька запитів одночасно. Під час тестів було перевірено, що бот працює ефективно навіть в умовах багатьох одночасних запитів. Оскільки запити до API виконуються асинхронно, це дозволяє боту обробляти їх без затримок, навіть якщо кількість користувачів збільшується.

6) Практичне застосування

Практичне застосування бота може значно полегшити користувачам процес пошуку авто, оскільки дозволяє отримувати оновлені оголошення без необхідності ручного переходу на сайт. Це особливо зручно для тих,

хто шукає конкретне авто, але хоче заощадити час на перегляді багатьох оголошень. Бот автоматизує пошук, дозволяючи користувачам швидко знаходити найвигідніші пропозиції в зручному форматі.

Також бот може бути корисний різним компаніям і маркетинговим агентствам, які займаються продажем або рекламою автомобілів. Він дозволяє отримувати інформацію про поточні пропозиції без необхідності ручного моніторингу ринку, що значно спрощує роботу з даними та підвищує ефективність бізнесу.

Telegram-бот, розроблений для пошуку автомобілів на основі API auto.ria.com, продемонстрував високу ефективність і простоту використання. Швидкість роботи, точність результатів пошуку, інтуїтивно зрозумілий інтерфейс і стабільність роботи під навантаженням роблять систему надійним інструментом для користувачів автопошуку. Крім того, система має великий потенціал для розширення та адаптації до інших видів товарів чи послуг, що дозволяє використовувати її в різних секторах бізнесу.

Загалом система є не лише корисним інструментом для кінцевих користувачів, а й ефективним рішенням для компаній, що займаються продажем автомобілів чи інших категорій товарів.

ВИСНОВОК

У результаті виконаної роботи на базі API сайту auto.ria.com розроблено Telegram-бот для пошуку автомобілів, який дозволяє користувачам швидко знаходити оголошення з автомобілями за заданими критеріями. Завдяки інтеграції з API та використанню бота Telegram система дозволяє автоматизувати процес пошуку та скоротити час збору інформації про автомобілі.

Вирішена проблема

Основною проблемою, яка була вирішена в рамках даної роботи, є створення зручного та ефективного інструменту пошуку автомобілів за допомогою сучасних технологій, зокрема месенджерів, за допомогою API зовнішнього сервісу. До цього часу більшість доступних рішень були зосереджені на використанні веб-сайтів або мобільних додатків для пошуку реклами. Однак використання таких платформ часто менш зручно, а інтерактивність і швидкість доступу до необхідної інформації залишає бажати кращого.

Значення для науки і практики

Робота має велике значення як для наукової, так і для практичної діяльності. У науковому контексті дослідження демонструє можливості інтеграції різних типів технологій (ботів Telegram і зовнішніх API) для автоматизації процесів, пов’язаних з обробкою та доставкою даних. Створення бота Telegram, який використовує API для отримання оновлених даних, є прикладом ефективного використання програмних засобів для полегшення щоденних завдань.

З практичної точки зору розробка такого бота має широке застосування для кінцевих користувачів, особливо людей, які шукають автомобілі, а також для компаній, що продають транспортні засоби. Бот дозволяє автоматизувати процес пошуку автомобіля та отримувати оновлену інформацію з різних джерел за допомогою єдиного інструменту, що значно спрощує вашу роботу та підвищує ефективність бізнесу.

Кількісні та якісні показники

Що стосується кількісних результатів, то в процесі тестування була досягнута висока швидкість обробки запитів. Час відповіді на запит користувача, що включає пошук та отримання даних з API, не перевищує 4 секунд. Це досить хороший результат для Telegram-ботів, які працюють із зовнішніми сервісами. Бот демонструє високу точність результатів, надаючи користувачам оновлені оголошення, які відповідають заданим параметрам.

Достовірність результатів

Достовірність отриманих результатів підтверджена методом тестування. Під час тестування були перевірені всі етапи роботи бота від ініціалізації пошуку до отримання результатів у вигляді реклами. Усі ці кроки було виконано правильно, а час обробки запиту відповідав очікуванням. Крім того, точність отриманих результатів підтверджується високим рівнем відповідності між результатами, отриманими через API, і даними, представленими на веб-сайті auto.ria.com.

Telegram-бот, розроблений для пошуку автомобілів, є ефективним і зручним інструментом, який забезпечує швидкий доступ до актуальних оголошень на основі API auto.ria.com. Всі завдання були успішно виконані, а розроблена система продемонструвала високу ефективність як для кінцевих споживачів, так і для компаній, що займаються продажем автомобілів. Отримані результати можуть значно покращити процес пошуку транспортних засобів, що робить цю систему важливим інструментом як для наукових досліджень, так і для практичного використання в реальному житті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Grigoryan, A., & Zorin, A. (2020). Telegram bots: Technology, development and applications. Springer.
2. Bender, H., & Sharma, D. (2018). APIs: A strategy guide. O'Reilly Media.
3. Microsoft. (2020). Bot Framework SDK for Python. – Retrieved from: https://learn.microsoft.com/en-us/azure/bot-service/
4. Asher, M. (2020). Working with APIs in Java. Packt Publishing.
5. Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley.
6. Salama, A., & Ghaleb, A. (2019). Telegram Bot Development: A practical approach to building bots using Node.js and Telegram API. Apress.
7. Wenzel, M. (2019). Learning REST APIs. Packt Publishing.
8. O'Reilly Media. (2021). Designing Web APIs: Building APIs That Developers Love. O'Reilly Media.
9. API Documentation - auto.ria.com. (2023). – Retrieved from: https://auto.ria.com/docs/api
10. BotMan. (2018). PHP Bot Framework. – Retrieved from: https://botman.io/
11. Telegram Bot API Documentation. (2021). – Retrieved from: https://core.telegram.org/bots/api
12. "Integrating Telegram Bots with REST APIs." (2019). Journal of Web Development, 15(2), 34-47.
13. Von Welter, S. (2020). Building Real-Time Web Apps with JavaScript and WebSockets. Packt Publishing.
14. Andreev, I. (2021). API for Beginners: Introduction to RESTful APIs. Udemy Online Course.
15. Johnson, R., & Hoeller, J. (2018). Spring in Action (5th Edition). Manning Publications.
16. Walls, C. (2019). Spring Boot in Action. Manning Publications.
17. Voskuil, M., & Mellinger, T. (2018). Spring Microservices in Action. Manning Publications.

18. Pivotal Software, Inc. (2020). Spring Boot Documentation. – Retrieved from: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
19. Bloch, J. (2018). Effective Java (3rd Edition). Addison-Wesley Professional.
20. Horstmann, C. S., & Cornell, G. (2018). Core Java Volume I: Fundamentals (11th Edition). Pearson.
21. Oracle. (2023). Java SE Documentation. – Retrieved from: https://docs.oracle.com/javase/
22. Schildt, H. (2018). Java: The Complete Reference (11th Edition). McGraw-Hill Education.
23. Fowler, M. (2010). Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional.
24. Johnson, R., & Webb, C. (2022). Pro Spring Boot 3: An Authoritative Guide to Building Java-Based Microservices and Applications. Apress.
25. Walls, C. (2021). Spring Boot in Action. Manning Publications.
26. Pivotal Software. (2023). Spring Boot Reference Documentation. – Retrieved from: https://docs.spring.io/spring-boot/docs/current/reference/html/
27. Heckler, M. (2021). Spring Boot: Up & Running: Building Cloud Native Java and Kotlin Applications. O'Reilly Media.
28. Tikal Technologies. (2023). Spring Boot Tutorial: Basics to Advanced. – Retrieved from: https://www.baeldung.com/spring-boot
29. Telegram Bot API Documentation. (2024). – Retrieved from: https://core.telegram.org/bots/api
30. Matveev, S. (2023). Building Telegram Bots with Java: A Step-by-Step Guide. O'Reilly Media.
31. Java Telegram Bot Library Documentation. (2024). – Retrieved from: https://github.com/rubenlagus/TelegramBots
32. Telegram Developers Blog. (2024). Best Practices for Bot Development. – Retrieved from: https://telegram.org/blog
33. BotFather. (2024). Creating and Managing Telegram Bots: Official Guide. – Retrieved from: https://core.telegram.org/bots/tutorial

34. MySQL Documentation. (2024). – Retrieved from: https://dev.mysql.com/doc/
35. DuBois, P. (2023). MySQL Cookbook: Solutions for Database Developers and Administrators. O'Reilly Media.
36. Schwartz, B., Zaitsev, P., & Tkachenko, V. (2022). High Performance MySQL: Optimization, Backups, Replication, and More. O'Reilly Media.
37. MySQL Performance Blog. (2024). – Retrieved from: https://www.percona.com/blog
38. DigitalOcean Tutorials. (2024). How to Use MySQL for Beginners. – Retrieved from: https://www.digitalocean.com/community/tutorials
39. Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Dissertation.
40. Richardson, L., Amundsen, M., & Ruby, S. (2022). RESTful Web APIs: Services for a Changing World. O'Reilly Media.
41. Postman Blog. (2024). Best Practices for API Design. – Retrieved from: https://blog.postman.com/
42. Swagger Documentation. (2024). – Retrieved from: https://swagger.io/docs/
43. Microsoft REST API Guidelines. (2024). – Retrieved from: https://github.com/microsoft/api-guidelines
44. JSON Official Website. (2024). Introducing JSON: The JavaScript Object Notation. – Retrieved from: https://www.json.org/
45. Bray, T. (2014). The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159.
46. JSON Schema Documentation. (2024). – Retrieved from: https://json-schema.org/
47. DigitalOcean Tutorials. (2024). Understanding JSON for Beginners. – Retrieved from: https://www.digitalocean.com/community/tutorials
48. Cunningham, W. (2023). JSON for Modern Data Interchange. O'Reilly Media.
49. Google Search Operators Guide. (2024). How to Use Advanced Search Commands. – Retrieved from: https://support.google.com/websearch/answer/2466433

50. Chowdhury, G. (2010). Introduction to Modern Information Retrieval. Facet Publishing.

Додаток А

Код для обробки запитів користувача

```
private void checkText(Update update, Long chatId, User user){
    String text = update.getMessage().getText();
    if(text.equals("Знайти оголошення")){
        user.setStatus(WAIT_CATEGORY);
        userService.save(user);
        sendKeyboardToChat("Оберіть тип транспорту:", chatId,
ReplyKeyboardMarkupUtils.categories());
    }
    else if(user.getStatus() == WAIT_CATEGORY){
        if(text.equals("Назад")){
            user.setStatus(UserStatus.FREE);
            userService.save(user);
            sendKeyboardToChat("Оберіть пункт меню:",
user.getTelegramId(), ReplyKeyboardMarkupUtils.menu());
        }
        else{
            TransportCategory category =
TransportCategory.fromDisplayName(text);
            if(category != null) {
user.setTempCategoryId(category.getCategoryId());
                user.setStatus(UserStatus.WAIT_BRAND);
                userService.save(user);
                sendKeyboardToChat("Введіть, будь ласка, марку
авто:", chatId, ReplyKeyboardMarkupUtils.back());
            }
            else
                sendMessageToChat("Такого типу транспорту не
існує, спробуйте ще раз:", chatId);
        }
    }
    else if(user.getStatus() == UserStatus.WAIT_BRAND){
        if(text.equals("Назад")){
            user.setStatus(UserStatus.WAIT_CATEGORY);
            userService.save(user);
            sendKeyboardToChat("Оберіть тип транспорту:", chatId,
ReplyKeyboardMarkupUtils.categories());
        }
        else{
            Integer brandId =
CarParametersSearch.searchBrand(text, user.getTempCategoryId());
            if(brandId != -1){
                user.setTempBrandId(brandId);
                user.setStatus(UserStatus.WAIT_MODEL);
                userService.save(user);
                sendKeyboardToChat("Тепер введіть модель авто:",
chatId, ReplyKeyboardMarkupUtils.back());
            }else
                sendMessageToChat("Такої марки авто не знайдено,
спробуйте ще раз:", chatId);
```

```
        }
    }
    else if(user.getStatus() == UserStatus.WAIT_MODEL){
        if(text.equals("Назад")){
            user.setStatus(UserStatus.WAIT_BRAND);
            userService.save(user);

            sendKeyboardToChat("Введіть, будь ласка, марку авто:",
chatId, ReplyKeyboardMarkupUtils.back());
        }
        else{
            new Thread(new Runnable() {
                @Override
                public void run() {
                    int modelId =
CarParametersSearch.searchModel(text, user.getTempBrandId(),
user.getTempCategoryId());
                    if(modelId != -1) {
                        List<String> ids =
CarParametersSearch.getCarIds(user.getTempCategoryId(),
user.getTempBrandId(), modelId);
                        if (!ids.isEmpty()) {
                            List<Car> cars =
CarParametersSearch.getCarsByIds(ids);
                            if (!cars.isEmpty()) {
                                for (Car car : cars) {
                                    String caption = "Назва: " +
car.getTitle() + "\nЦіна: " + car.getPrice() + " грн.\n" +
"Розташування: " + car.getCity() +
                                                    "\nОпис: \n" +
car.getDescription().trim() + "\n\nПосилання: " + car.getLink();
sendPhotoByUrlToChat(car.getImgLink(), chatId, caption);
                                    try {
                                        Thread.sleep(1500);
                                    } catch (Exception ex) {
                    ex.printStackTrace();
                                    }
                                }
                            } else {
                                user.setStatus(FREE);
                                userService.save(user);

                                sendKeyboardToChat("На жаль, за
введеними параметрами не знайдено жодного авто.", chatId,
ReplyKeyboardMarkupUtils.menu());
                            }
                        } else {
                            user.setStatus(FREE);
                            userService.save(user);

                            sendKeyboardToChat("На жаль, за
введеними параметрами не знайдено жодного авто.", chatId,
ReplyKeyboardMarkupUtils.menu());
```

```
                        }
                    }else
                        sendMessageToChat("Такої моделі не
знайдено, спробуйте іншу.", chatId);

                }
            }).start();
        }
    }
}
```

Додаток Б

Код класу, що здійснює роботу з API Auto.RIA

```
public class CarParametersSearch {
private static final String API_KEY = "";
    private static final HttpClient httpClient =
HttpClient.newHttpClient();
    private static final ObjectMapper objectMapper = new
ObjectMapper();
    private static List<CarBrand> fetchCarBrands(int categoryId)
throws IOException, InterruptedException {
        String url = "https://developers.ria.com/auto/categories/"
+ categoryId + "/marks?api_key=" + API_KEY;
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(url))
                .GET()
                .build();
        HttpResponse<String> response = httpClient.send(request,
HttpResponse.BodyHandlers.ofString());
        if (response.statusCode() != 200) {
            throw new RuntimeException("Failed to fetch car
brands: HTTP status code " + response.statusCode());
        }
        return objectMapper.readValue(response.body(), new
TypeReference<List<CarBrand>>() {});
    }
    private static List<CarModel> fetchCarModel(int brandId, int
categoryId) throws IOException, InterruptedException {
        String url = "https://developers.ria.com/auto/categories/"
+ categoryId + "/marks/" + brandId + "/models?api_key=" + API_KEY;
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(url))
                .GET()
                .build();
        HttpResponse<String> response = httpClient.send(request,
HttpResponse.BodyHandlers.ofString());
        if (response.statusCode() != 200) {
            throw new RuntimeException("Failed to fetch car
brands: HTTP status code " + response.statusCode());
        }
        return objectMapper.readValue(response.body(), new
TypeReference<List<CarModel>>() {});
    }
    public static Integer searchBrand(String brand, int
categoryId){
        try {
            List<CarBrand> brands = fetchCarBrands(categoryId);
            if(!brands.isEmpty()){
                for (CarBrand carBrand : brands) {

if(carBrand.getName().equalsIgnoreCase(brand.trim()))
                        return carBrand.getValue();
```

```
                }
            }
            else
                return -1;
        }catch (Exception ex){
            ex.printStackTrace();
        }
        return -1;
    }
    public static Integer searchModel(String model, int brandId,
int categoryId){
        try {
            List<CarModel> models = fetchCarModel(brandId,
categoryId);
            if(!models.isEmpty()){
                for (CarModel carModel : models) {
                    if(carModel.getName().equalsIgnoreCase(model))
                        return carModel.getValue();
                }
            }
            else
                return -1;
        }catch (Exception ex){
            ex.printStackTrace();
        }
        return -1;
    }
    public static List<String> getCarIds(int categoryId, int
brandId, int modelId){
        String url =
"https://developers.ria.com/auto/search?api_key=" + API_KEY +
"&category_id=" + categoryId + "&marka_id=" + brandId +
"&model_id=" + modelId;
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(url))
                .GET()
                .build();
        List<String> ids = new ArrayList<>();
        try {
            HttpResponse<String> response =
httpClient.send(request, HttpResponse.BodyHandlers.ofString());
            JSONObject jsonObject = new
JSONObject(response.body());
            if(jsonObject.has("result")){
                JSONObject resultObject =
jsonObject.getJSONObject("result");
                if(resultObject.has("search_result")){
                    JSONObject searchResultObject =
resultObject.getJSONObject("search_result");
                    if(searchResultObject.has("ids")){
                        JSONArray idsArray =
searchResultObject.getJSONArray("ids");
                        for (int i = 0; i < idsArray.length();
```

```
i++)
                                ids.add(idsArray.getString(i));
                        }
                    }
                }
            }catch (Exception ex){
                ex.printStackTrace();
            }
            return ids;
        }
        public static List<Car> getCarsByIds(List<String> ids){
            String url =
"https://developers.ria.com/auto/info?api_key=" + API_KEY +
"&auto_id=";
            List<Car> cars = new ArrayList<>();
            for (String id : ids) {
                try{
                    HttpRequest request = HttpRequest.newBuilder()
                            .uri(URI.create(url + id))
                            .GET()
                            .build();
                    HttpResponse<String> response =
httpClient.send(request, HttpResponse.BodyHandlers.ofString());
                    JSONObject jsonObject = new
JSONObject(response.body());
                    Car car = new Car();
                    if (jsonObject.has("prices")) {
                        JSONArray pricesArray =
jsonObject.getJSONArray("prices");
                        if (!pricesArray.isEmpty()) {
                            JSONObject firstPriceObject =
pricesArray.getJSONObject(0);
                            if (firstPriceObject.has("UAH")) {
                                String price =
firstPriceObject.getString("UAH");
                                price = price.replace(" ", ".");

                                car.setPrice(price);
                            }
                        }
                    }
                    if(jsonObject.has("title")){
                        String title = jsonObject.getString("title");
                        car.setTitle(title);
                    }
                    if(jsonObject.has("linkToView")){
                        String link = "https://auto.ria.com" +
jsonObject.getString("linkToView");
                        car.setLink(link);
                    }
                    if(jsonObject.has("cityLocative")){
                        String city = "y " +
jsonObject.getString("cityLocative");
```

```
                    car.setCity(city);
                }
                if(jsonObject.has("autoData")){
                    JSONObject autoDataObject =
jsonObject.getJSONObject("autoData");
                    String description =
autoDataObject.getString("description");
                    car.setDescription(description);
                }
                if(jsonObject.has("photoData")){
                    JSONObject photoDataObject =
jsonObject.getJSONObject("photoData");
                    String photoLink =
photoDataObject.getString("seoLinkF");
                    car.setImgLink(photoLink);
                }
                cars.add(car);
            }catch (Exception ex){
                ex.printStackTrace();
            }
        }return cars;}}
```