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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка телеграм-бота на мові програмування Java для автоматизованого пошуку вакансій і генерації супровідного листа»

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

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

_____ Володимир Большаков
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Володимир Большаков
група КНД-41

Керівник: к.т.н., доцент Іщеряков Сергій Михайлович
науковий ступінь, вчене звання

Рецензент: _____
науковий ступінь, вчене звання *(Ім'я, ПРІЗВИЩЕ)*

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Бакалавр

Спеціальність Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Большакову Володимиру Руслановичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Розробка телеграм-бота на мові програмування Java для автоматизованого пошуку вакансій і генерації супровідного листа»

керівник кваліфікаційної роботи Сергій ІЩЕРЯКОВ к.т.н., доцент,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» 02.2024р. №36

2. Строк подання кваліфікаційної роботи «31» травня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з питань, що пов'язані з розробкою за допомогою мови Java

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз існуючих методів автоматизованого пошуку вакансій

Розробка алгоритмів для пошуку вакансій та генерації супровідних листів

Створення телеграм-бота на мові Java, який використовує створені алгоритми

Протестувати телеграм-бота та оцінити його ефективність

5. Перелік графічного матеріалу: *презентація*

1. Актуальність роботи
2. Кількість активних користувачів Telegram
3. Боти для пошуку роботи
4. Інструменти розробки
5. Провайдер вакансій
6. Породжувальний штучний інтелект
7. Експлуатація бота
8. Експлуатація бота
9. Висновки
10. Апробація

6. Дата видачі завдання «23» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	23.02 – 01.03	
2	Аналіз доступних бібліотек для розробки	02.03 – 16.03	
3	Створення скелету застосунку	16.03 – 24.03	
4	Розробка застосунку	25.03 – 11.04	
5	Тестування та налагодження застосунку	12.04 – 26.04	
6	Написання вступу, висновків, реферату	27.04 – 08.05	
7	Написання основних розділів пояснювальної записки	09.05 – 30.05	
8	Подання роботи в деканат	31.05	

Здобувач вищої освіти _____ Володимир БОЛЬШАКОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Сергій ІЩЕРЯКОВ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 34с., 15 джерел.

Наукове завдання – проаналізувати існуючі методи автоматизованого пошуку вакансій та генерації супровідних листів; розробити алгоритми для пошуку вакансій та генерації супровідних листів; створити телеграм-бота на мові програмування Java, який використовує розроблені алгоритми; провести тестування телеграм-бота та оцінити його ефективність.

Мета роботи – розробка телеграм-бота, який допомагає користувачам знаходити релевантні вакансії та економити час на написанні супровідних листів.

Об'єкт дослідження – автоматизований пошук вакансій та генерація супровідного листа за допомогою телеграм-бота, розробленого на мові програмування Java.

Предмет дослідження – програма-телеграм-бот, що використовує Java для автоматизованого пошуку вакансій та породжувальний штучний інтелект для генерації супровідних листів.

Короткий зміст роботи: У роботі були використані сучасні підходи до розробки застосунків на мові програмування Java. Для генерації супровідних листів був використаний породжувальний штучний інтелект від OpenAI ChatGPT 3.5 Turbo 0125.

Даний застосунок може бути використаний кандидатами які зацікавлені в інструменті для автоматизованого пошуку вакансій та генерації супровідних листів.

КЛЮЧОВІ СЛОВА: ПОРОДЖУВАЛЬНИЙ ШТУЧНИЙ ІНТЕЛЕКТ, ТЕЛЕГРАМ-БОТ, JAVA, ПОШУК ВАКАНСІЙ, СУПРОВІДНИЙ ЛИСТ

ABSTRACT

The part of the bachelor qualification work: 34 pages, 15 sources.

Research task - analyze existing methods for automated job search and cover letter generation; develop algorithms for job search and cover letter generation; create a Telegram bot using the Java programming language that utilizes the developed algorithms; test the Telegram bot and evaluate its effectiveness.

Goal - develop a Telegram bot that assists users in finding relevant job openings and saves time writing cover letters.

Research subject - automated job search and cover letter generation using a Java-developed Telegram bot.

Research object - a Telegram bot program that utilizes Java for automated job search and generative AI for cover letter generation.

Summary: The research employed modern approaches to developing Java applications. OpenAI ChatGPT 3.5 Turbo 0125 generative AI was used for cover letter generation.

This application can be used by candidates seeking a tool for automated job search and cover letter generation.

KEYWORDS**:** GENERATIVE AI, TELEGRAM BOT, JAVA, JOB SEARCH, COVER LETTER

ЗМІСТ

ВСТУП

В сучасному світі ринок праці динамічно розвивається, а пошук роботи стає все більш складним завданням. Кандидатам на вакансії необхідно витрачати багато часу та зусиль, щоб знайти відповідні пропозиції, скласти резюме та супровідні листи. Ці завдання можуть бути рутинними та виснажливими, що негативно впливає на мотивацію та шанси на успішне працевлаштування.

Впровадження штучного інтелекту та автоматизації в сфері пошуку роботи може значно полегшити та оптимізувати цей процес. Телеграм-боти, створені на основі мови програмування Java, є одним з перспективних інструментів для автоматизації завдань, пов'язаних з пошуком вакансій та генерацією супровідних листів.

Актуальність даної теми дослідження обумовлюється наступними факторами:

- Зростання популярності телеграм-ботів: Телеграм є однією з найпопулярніших платформ для обміну повідомленнями, яка має мільйони активних користувачів. Використання телеграм-ботів для автоматизації завдань стає все більш поширеним, адже це зручний та доступний спосіб отримати необхідну інформацію та послуги[7].

- Ефективність автоматизації: Автоматизовані телеграм-боти можуть значно скоротити час та зусилля, які витрачають кандидати на пошук вакансій та написання супровідних листів. Це дозволяє їм зосередитися на більш важливих аспектах працевлаштування, таких як підготовка до співбесіди та розвиток професійних навичок.

- Персоналізація супровідних листів: Автоматизовані телеграм-боти можуть генерувати персоналізовані супровідні листи, які враховують досвід, навички та

інтереси кандидата, а також специфіку вакансії. Це значно підвищує шанси на успішне проходження співбесіди та отримання бажаної роботи.

- Доступність інформації: Телеграм-боти можуть надавати доступ до широкого спектру інформації про вакансії, включаючи опис посадових обов'язків, вимоги до кандидатів, контактну інформацію роботодавця та відгуки про компанію. Це дозволяє кандидатам зробити більш обґрунтований вибір при пошуку роботи.

Використання телеграм-ботів для автоматизованого пошуку вакансій та генерації супровідних листів може стати цінним інструментом для кандидатів на працевлаштування, роботодавців та кадрових агентств. Ця технологія має потенціал значно оптимізувати процес пошуку роботи та зробити його більш ефективним для всіх учасників.

Мета дослідження:

- Розробити телеграм-бота на мові програмування Java для автоматизованого пошуку вакансій та генерації супровідного листа.
- Дослідити можливості та перспективи використання телеграм-ботів для оптимізації процесу пошуку роботи.
- Оцінити ефективність телеграм-бота в реальних умовах використання.

Завдання дослідження:

- Проаналізувати існуючі телеграм-боти для пошуку вакансій.
- Визначити функціональні можливості та характеристики телеграм-бота, який буде розроблений.
- Розробити програмне забезпечення телеграм-бота на мові програмування Java.
- Здійснити тестування та налагодження телеграм-бота.

Очікувані результати дослідження:

- Розроблений та протестований телеграм-бот на мові програмування Java для автоматизованого пошуку вакансій та генерації супровідного листа.
- Аналітичний опис функціональних можливостей та характеристик телеграм-бота.
- Рекомендації щодо вдосконалення та подальшого розвитку телеграм-бота.

Практична значимість дослідження:

- Розроблений телеграм-бот може бути використаний для автоматизованого пошуку вакансій та генерації супровідних листів.
- Результати дослідження можуть бути корисними для розробників телеграм-ботів, роботодавців та HR агентств.
- Дослідження може сприяти розвитку нових технологій та інструментів для автоматизації процесу пошуку роботи.

1 ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ТЕЛЕГРАМ-БОТА

1.1 Поняття та характеристики телеграм-ботів

Телеграм-бот – це програмне забезпечення, що функціонує в межах месенджера Telegram та здатне виконувати широкий спектр завдань[2].

На відміну від звичайних користувачів, телеграм-боти мають унікальний ідентифікатор (токен), який використовується для їх авторизації та доступу до API Telegram. Цей інтерфейс прикладного програмування надає розробникам набір команд та методів для керування ботом, його взаємодії з користувачами та виконання різноманітних дій.

До ключових характеристик телеграм-ботів належать:

- Інтерактивність: Пряма комунікація з користувачами в режимі реального часу. Бот може відповідати на повідомлення, надсилати текстові та голосові повідомлення, фотографії, відео, стикери, геолокаційні дані та інші типи контенту.

- Функціональність: Широкий спектр виконуваних завдань. Бот може виконувати дії за командами користувачів, інтегруватися з іншими сервісами, обробляти інформацію, генерувати контент та багато іншого.

- Простота використання: Зручність та доступність для користувачів без спеціальних знань чи навичок. Користувачі взаємодіють з ботом, надсилаючи йому повідомлення та використовуючи команди, що робить його використання інтуїтивно зрозумілим.

- Безкоштовність: Доступність телеграм-ботів безкоштовно для всіх користувачів Telegram. Це робить їх доступними для широкої аудиторії та стимулює їхнє поширення.

- Широке поширення: Мільйонна аудиторія Telegram забезпечує охоплення широкої цільової аудиторії. Більшість користувачів Telegram знайомі з концепцією ботів, що полегшує їхнє впровадження та використання.

Переваги використання телеграм-ботів:

- Економія часу: Автоматизація рутинних завдань, що економить час та ресурси користувачів. Бот може виконувати повторювані дії значно швидше та ефективніше, ніж людина.

- Зручність: Доступність телеграм-ботів у будь-який час та з будь-якого місця, де є доступ до інтернету. Користувачі можуть взаємодіяти з ботом за допомогою своїх смартфонів, планшетів або комп'ютерів.

- Персоналізація: Можливість персоналізації телеграм-ботів відповідно до потреб та інтересів кожного користувача. Бот може адаптувати свою поведінку та надавати інформацію, що відповідає індивідуальним потребам користувача.

- Інтеграція: Широкі можливості інтеграції з іншими сервісами та платформами. Бот може підключатися до різних веб-сервісів, API та баз даних, що розширює його функціональність та можливості.

Сфера застосування телеграм-ботів охоплює:

- Інформаційні послуги: Надання інформації про погоду, новини, курси валют, розклади транспорту тощо. Бот може бути джерелом актуальної та корисної інформації для користувачів.

- Обслуговування клієнтів: Відповідання на запитання, надання підтримки та вирішення проблем клієнтів. Бот може слугувати віртуальним помічником, який забезпечує швидке та якісне обслуговування клієнтів.

- Освіта: Використання для навчання, тестування та оцінювання знань. Бот може бути інтерактивним інструментом для вивчення різних предметів та навичок.

- Розваги: Забезпечення ігор, вікторин, розваг та інших видів дозвілля. Бот може слугувати джерелом розваг та емоцій для користувачів.

- Електронна комерція: Здійснення продажу товарів та послуг. Бот може приймати замовлення, обробляти платежі

1.2 Архітектура телеграм-ботів

Архітектура телеграм-бота являє собою структуру його компонентів та їх взаємодію між собою. Вона визначає принцип роботи бота, його функціональність та можливості.

Основні компоненти архітектури телеграм-бота:

API Telegram: Інтерфейс прикладного програмування, який надає розробникам набір команд та методів для керування ботом, його взаємодії з користувачами та виконання різноманітних дій.[12]

Сервер бота: Програмне забезпечення, що розміщене на сервері та відповідає за обробку запитів від Telegram API, виконання логіки роботи бота та взаємодію з користувачами.

База даних: Місце зберігання даних, необхідних для роботи бота, таких як інформація про користувачів, налаштування бота, дані для виконання завдань тощо.

Інтерфейс користувача: Спосіб взаємодії користувачів з ботом, як правило, через текстові команди або графічний інтерфейс.

Моделі архітектури телеграм-ботів:

Однокомпонентна: Всі компоненти бота розміщені на одному сервері. Це проста та економна модель, але вона може мати обмеження щодо масштабованості та продуктивності.

Багатокомпонентна: Різні компоненти бота розміщені на різних серверах. Це більш складна модель, але вона забезпечує кращу масштабованість, продуктивність та надійність.

Мікросервісна: Бот розбитий на дрібні незалежні мікросервіси, кожен з яких виконує певну функцію. Це гнучка та масштабована модель, але вона потребує більш складного розгортання та обслуговування.

Вибір архітектури телеграм-бота залежить від:

Функціональності бота: Чим складніші функції виконує бот, тим більш складна архітектура йому потрібна.

Очікуваного навантаження: Якщо очікується, що бот буде мати багато користувачів, йому потрібна масштабована архітектура, яка може витримувати високе навантаження.

Бюджету та ресурсів: Розробка та обслуговування більш складної архітектури потребує більших ресурсів.

Важливо ретельно продумати архітектуру телеграм-бота перед його розробкою, щоб вона відповідала його потребам та можливостям.

1.3 Мови програмування для розробки телеграм-ботів

Вибір мови програмування для розробки телеграм-бота залежить від декількох факторів:

Функціональність бота:

- Для простих ботів, що виконують базові завдання, може підійти Python або JavaScript.

- Для складніших ботів, що потребують інтеграції з іншими сервісами та обробки великих обсягів даних, може знадобитися Java, C++ або Go.

Досвід розробника:

- Якщо ви володієте певною мовою програмування, то краще використовувати саме її, адже це значно прискорить процес розробки.

- Існують також платформи, бібліотеки та фреймворки, що полегшують розробку телеграм-ботів, не потребуючи глибоких знань програмування.

Популярність мови:

- Деякі мови програмування мають більшу спільноту розробників телеграм-ботів, що може полегшити пошук інформації та допомоги.

Ось деякі з найпопулярніших мов програмування для розробки телеграм-ботів:

- Python: Проста та універсальна мова програмування з великою спільнотою розробників. Існує багато бібліотек та фреймворків для розробки телеграм-ботів на Python, таких як telepot, Python-telegram-bot та aiogram.

- JavaScript: Ще одна популярна та універсальна мова програмування з великою спільнотою розробників. Існує також багато бібліотек та фреймворків для розробки телеграм-ботів на JavaScript, таких як Node Telegram Bot API, Telegraf та Botkit.

- Java: Потужна та масштабована мова програмування, що добре підходить для складних ботів. Існує багато бібліотек та фреймворків для розробки телеграм-ботів на Java, таких як Java Telegram Bot API та Telegram Bots.

- C++: Швидка та ефективна мова програмування, що може бути корисною для ботів, які потребують високої продуктивності. Існує декілька бібліотек для розробки телеграм-ботів на C++, такі як tgbot-cpp та tgbot.

- Go: Сучасна та масштабована мова програмування, що стає все більш популярною для розробки телеграм-ботів. Існує декілька бібліотек для розробки телеграм-ботів на Go, таких як go-telegram-bot-api та go-telegram/bot.

1.4 Інструменти та платформи для розробки телеграм-ботів

Існує безліч інструментів та платформ, які можуть полегшити розробку телеграм-ботів та зробити її більш доступною для користувачів з різним рівнем досвіду програмування.

Ось деякі з найпопулярніших інструментів та платформ:

- BotFather: Офіційний інструмент від Telegram для створення та керування ботами. BotFather дозволяє створити бота, отримати його токен та налаштувати базові параметри[13].

- Manybot: Платформа для створення ботів без необхідності програмування. Manybot пропонує візуальний конструктор, який дозволяє створювати ботів з drag-and-drop елементами[6].

- Rasa: Платформа для створення чат-ботів з штучним інтелектом. Rasa дозволяє створювати ботів, які можуть розуміти природну мову та вести діалог з користувачами[9].

- Flow XO: Платформа для автоматизації робочих процесів, яка також може використовуватися для розробки телеграм-ботів. Flow XO пропонує потужні інструменти для автоматизації завдань, інтеграції з іншими сервісами та аналітики даних[4].

- Dialogflow: Хмарна платформа для розробки чат-ботів з відкритим API, яка також може використовуватися для створення телеграм-ботів. Dialogflow пропонує візуальний конструктор ботів, інтеграцію з Google Cloud Platform та інші функції[3].

Вибір інструменту або платформи для розробки телеграм-бота залежить від:

Ваших навичок програмування: Якщо ви не маєте досвіду програмування, то краще використовувати платформу з візуальним конструктором ботів.

Функціональності бота: Деякі інструменти та платформи краще підходять для створення простих ботів, а інші – для складних ботів з машинним навчанням та обробкою природної мови.

Вашого бюджету: Деякі інструменти та платформи є безкоштовними, а інші потребують платної підписки.

Важливо зазначити, що використання інструментів та платформ не гарантує створення якісного телеграм-бота. Для цього все ще потрібні базові знання принципів роботи телеграм-ботів та логіки програмування.

Окрім цих платформ, існує безліч бібліотек та фреймворків для розробки телеграм-ботів на різних мовах програмування. Ці бібліотеки та фреймворки надають готові компоненти та функції, які полегшують розробку та тестування ботів.

Вибір інструмента або платформи для розробки телеграм-бота залежить від ваших потреб, досвіду та бюджету. Якщо вам потрібна проста та швидка розробка бота, то платформа без програмування може бути гарним вибором. Якщо ж вам потрібна більш гнучка та функціональна розробка, то вам може знадобитися використовувати бібліотеку або фреймворк на мові програмування, з якою ви володієте.

1.5 Огляд існуючих телеграм-ботів для пошуку вакансій

Існує безліч телеграм-ботів, які допомагають користувачам шукати вакансії. Ці боти пропонують різні функції, такі як:

- Пошук вакансій за ключовими словами: Користувачі можуть ввести ключові слова, що описують бажану посаду, галузь або місце розташування, щоб знайти відповідні вакансії.

- Підписка на вакансії: Користувачі можуть підписатися на отримання нових вакансій, які відповідають їхнім критеріям пошуку.

- Пряме спілкування з роботодавцями: Деякі боти дозволяють користувачам напрямую спілкуватися з роботодавцями, щоб дізнатися більше про вакансії та подати заявку.

- Підготовка резюме: Деякі боти пропонують інструменти для створення та редагування резюме.

- Підготовка до співбесіди: Деякі боти пропонують поради та ресурси для підготовки до співбесіди.

Ось декілька популярних телеграм-ботів для пошуку вакансій:

- Work UA Bot: Цей бот пропонує широкий спектр функцій для пошуку вакансій в Україні. Користувачі можуть шукати вакансії за ключовими словами, підписуватися на вакансії, спілкуватися з роботодавцями та отримувати допомогу з підготовкою резюме та до співбесіди[14].

- Djinni Bot: Цей бот спеціалізується на пошуку вакансій в IT-сфері. Користувачі можуть шукати вакансії за мовами програмування, технологіями та рівнем досвіду[15].

- Rabota.ua Bot: Цей бот є офіційним ботом сайту з пошуку роботи robota.ua. Користувачі можуть шукати вакансії з бази сайту, підписуватися на вакансії та отримувати сповіщення про нові вакансії[10].

- Jooble Bot: Цей бот пропонує пошук вакансій з сайту Jooble [5].

Важливо зазначити, що більшість існуючих телеграм-ботів для пошуку вакансій не надають функціоналу для автоматичного генерування супровідного листа відповідно до опису вакансії та опису про себе, який надав користувач бота.

Це означає, що користувачам все ще потрібно самостійно складати супровідні листи, щоб відповісти на конкретні вимоги вакансії.

Деякі боти можуть запропонувати шаблони супровідних листів або підказки, які допоможуть користувачам розпочати роботу.

Однак, ці шаблони та підказки не замінюють повністю самостійного написання супровідного листа.

Це створює нішу для розробки бота, який зможе генерувати супровідні листи, полегшуючи та покращуючи процес пошуку роботи для користувачів.

Відсутність функціоналу для автоматичного генерування супровідних листів у телеграм-ботах для пошуку вакансій має декілька причин:

- Складність завдання: Створення супровідного листа, який ефективно підкреслює кваліфікацію та досвід користувача, є складним завданням, яке потребує розуміння вакансії та вміння чітко та лаконічно висловлювати свої думки.

- Індивідуальність: Кожна вакансія має свої унікальні вимоги, тому супровідний лист, який підходить для однієї вакансії, може не підійти для іншої.

- Відповідальність: Складання супровідного листа є важливою частиною процесу пошуку роботи, і користувач несе відповідальність за те, щоб лист був якісним та відповідав вимогам вакансії.

Незважаючи на відсутність функціоналу для автоматичного генерування супровідних листів, телеграм-боти для пошуку вакансій можуть бути цінним інструментом для полегшення процесу пошуку роботи.

Ці боти можуть допомогти користувачам знайти відповідні вакансії, підписатися на сповіщення про нові вакансії та спілкуватися з роботодавцями.

Користувачі, які шукають роботу, можуть використовувати телеграм-боти для пошуку вакансій, але їм все ще потрібно буде самостійно скласти супровідні листи, щоб збільшити свої шанси на успіх.

Розробка телеграм-бота з функціоналом автоматичного генерування супровідних листів може допомогти користувачам:

- Економити час та зусилля: Генерування супровідного листа може бути трудомістким процесом, а бот може автоматизувати цю задачу, звільняючи час для інших аспектів пошуку роботи.

- Збільшити шанси на успіх: Персоналізований та добре написаний супровідний лист може значно збільшити шанси користувача на отримання запрошення на співбесіду.

- Підвищити впевненість: Користувачі, які не мають досвіду написання супровідних листів, можуть відчувати себе більш впевнено, знаючи, що бот може допомогти їм створити якісний супровідний лист.

2 РОЗРОБКА ТЕЛЕГРАМ-БОТА НА МОВІ ПРОГРАМУВАННЯ JAVA

2.1 Вибір інструментів та бібліотек для розробки

Для розробки телеграм-бота на мові програмування Java були обрані наступні інструменти та платформи:

Інтегроване середовище розробки (IDE):

- IntelliJ IDEA: Це потужне IDE пропонує широкий спектр функцій для розробки Java, включаючи автодоповнення коду, рефакторинг, відладку та інтеграцію з системами контролю версій.

Версія Java:

- Java 17: Я буду використовувати передостанню LTS версію Java, яка пропонує нові функції та покращення, такі як удосконалені вирази switch, records та інструменти для аналізу коду.

Фреймворки та бібліотеки:

- Spring Boot 3: Цей популярний фреймворк Spring спрощує розробку веб-застосунків та мікросервісів Java. Він пропонує автоконфігурацію, залежності вбудовані в Spring, та інструменти для тестування та розгортання.

- Spring Batch: Цей фреймворк Spring надає інструменти для обробки пакетних даних, що може бути корисно для таких завдань, як імпорт даних про вакансії або обробка великих обсягів користувацьких даних.

- Spring Data JPA: Ця бібліотека спрощує доступ та роботу з базами даних JPA. Вона дозволяє легко зберігати, отримувати та оновлювати дані про вакансії, користувачів та інші сутності бота.

- Spring AI: Цей фреймворк Spring надає інструменти для машинного навчання та штучного інтелекту, які можна використовувати для автоматичного генерування супровідних листів.

- Liquibase: Цей інструмент управління базами даних допомагає утримувати структуру бази даних в актуальному стані, застосовуючи зміни контрольованим способом.

- PostgreSQL: Ця вільна та відкрита реляційна база даних пропонує високу продуктивність, масштабованість та надійність, що робить її гарним вибором для зберігання даних бота.

Бібліотека для взаємодії з Telegram API:

- java-telegram-bot-api (com.github.pengrad): Ця популярна бібліотека надає простий та зручний інтерфейс для взаємодії з Telegram API, що дозволяє боту надсилати та отримувати повідомлення, обробляти команди та керувати користувачами.

Інші інструменти:

- Jsoup: Ця бібліотека Java для роботи з HTML та XML може бути корисною для обробки веб-сторінок з вакансіями або інших джерел даних.

2.2 Реалізація функціоналу пошуку вакансій

Пошук вакансій для кожного сайту буде відрізнятися, тому потрібно створити спільний інтерфейс який буде реалізований кожним сервісом окремо. Так як задумка бота шукати вакансії кожні 30-60 хвилин, метод повинен приймати масив ключових слів які цікавлять користувача для пошуку та час останього пошуку, щоб відсіювати вакансії які вже були оброблені. Крім методу для пошуку вакансій, треба створити метод який буде повертати кількість вакансій для певного набору ключових слів. Перед додаванням пошуку до системи бота, користувач

повинен бачити кількість доступних вакансій, та подумати чи він хоче мати цей пошук увімкненим для оновлення та генерації супровідних листів.

Наш інтерфейс для пошуку вакансій буде називатися VacancyProviderService та мати такий список методів:

```
1 Vacancies getLastVacancies(List<String> searchKeywords, Instant from);
2 int getNumberOfVacancies(List<String> searchKeywords);
3 VacancyProvider getProvider();
4 String getUrl(List<String> searchKeywords);
5 String getUrl();
```

У майбутньому пошук вакансій буде доступний для великої кількості сайтів. Щоб сервісу який реагує на відправлені команди користувача не довелось мати 5+ та більше сервісів у собі, потрібно реалізувати один сервіс «фасад» який буде приймати не тільки параметри з ключовими словами для пошуку, та часом останього пошуку, а й значення провайдеру вакансій для якого цей пошук треба здійснити.

Для вирішення цієї задачі застосунок має окремний інтерфейс – VacancyProvidersService з такими методами всередині:

```
1 Vacancies getLastVacancies(VacancyProvider provider, List<String> searchKeywords, Instant from);
2 int getNumberOfVacancies(VacancyProvider provider, List<String> searchKeywords);
```

Сервіс який буде реалізовувати цей інтерфейс, може без проблем отримати за допомогою Dependency Injection від Spring Boot всі сервіси інтерфейсу VacancyProviderService у масиві та зібрати їх у словник EnumMap<VacancyProvider, VacancyProviderService> та просто виступати як proxy

між сервісом для отримання вакансій та сервісом який зацікавлений у отримання нових вакансій.

2.2.1 Підключення до API пошукових систем вакансій

Багато сайтів з пошуку вакансій пропонують REST API, який дозволяє отримувати доступ до даних про вакансії за допомогою програмних запитів. Цей API зазвичай використовує формат JSON або XML для передачі даних. Для роботи з REST API зазвичай потрібна реєстрація для отримання ключа API, який використовується для аутентифікації запитів. Переваги використання REST API:

Простота використання: REST API зазвичай мають просту та зрозумілу документацію, що робить їх легкими для інтеграції з програмами.

Гнучкість: REST API дозволяють отримувати доступ до даних у різних форматах, таких як JSON або XML.

Масштабованість: REST API можуть обробляти велику кількість запитів, що робить їх придатними для використання в масштабних програмах.

Зараз для реалізації HTTP запитів стандартою бібліотекою у світі Java розробки виступає Feign.

Feign - це бібліотека Java, яка використовується для спрощення декларативного визначення та виклику HTTP-запитів. Feign використовує аннотації для визначення методів, які відповідають HTTP-запитам, і автоматично генерує код для виконання цих запитів. Переваги використання Feign:

Простота використання: Feign дозволяє декларативно визначати HTTP-запити за допомогою аннотацій, що робить їх легкими для використання.

Абстрагування: Feign абстрагує деталі реалізації HTTP-запитів, що дозволяє розробникам зосередитися на бізнес-логіці своїх програм.

Підтримка різних протоколів: Feign підтримує різні протоколи HTTP, такі як GET, POST, PUT і DELETE.

У нашому випадку ми будемо отримувати вакансії від Djinni який не має API. Тому для отримання вакансій ми будемо користуватися іншою бібліотекою та парсити значення самостійно.

У деяких випадках, коли доступ до API пошукової системи вакансій неможливий або складний, можна використовувати бібліотеку для парсингу HTML Jsoup. Jsoup дозволяє парсити HTML-документи та витягувати з них дані. Це може бути корисно для отримання даних про вакансії з сайтів, які не пропонують API. Переваги використання Jsoup:

Простота використання: Jsoup має простий і зрозумілий API, що робить його легким для використання.

Гнучкість: Jsoup дозволяє парсити HTML-документи будь-якої складності.

Потужність: Jsoup пропонує широкий спектр функцій для парсингу HTML-документів.

Недоліки використання Jsoup:

Складність: Парсинг HTML-документів може бути складним завданням, особливо якщо структура документів складна.

Необхідність оновлення: Структура HTML-документів може змінюватися з часом, що може призвести до необхідності оновлення коду парсингу.

2.2.2 Обробка та фільтрація результатів пошуку

Для отримання вакансій сайту Djinni нам для початку потрібно ознайомитися зі структурою HTML сторінки яку ми отримуємо. Є багато способів щоб це зробити, така функція доступна по замовчуванню у всіх сучасних веб-браузерах. Також можна використовувати спеціалізовані застосунки які створені для виконання HTTP запитів, отримання результатів запиту та подальшого аналізу. Я для цих цілей буду використовувати Postman.

Postman - це популярна платформа для розробки API, яка пропонує широкий спектр функцій для роботи з HTTP-запитами та відповідями[8].

Можливості Postman, які можуть бути корисними для аналізу HTML-сторінок:

- Відправлення HTTP-запитів: Postman дозволяє надсилати HTTP-запити до веб-серверів і отримувати відповіді. Це може бути корисно для отримання HTML-сторінок, які потрібно проаналізувати.
- Перегляд відповідей: Postman дозволяє переглядати відповіді HTTP, включаючи HTML-код. Це може бути корисно для розуміння структури HTML-сторінки.
- Витяг даних: Postman дозволяє витягувати дані з HTML-відповідей. Це може бути корисно для отримання інформації, такої як текст, посилання та зображення.
- Створення колекцій: Postman дозволяє створювати колекції запитів, які можна використовувати для групування та повторного використання. Це може бути корисно для організації запитів, які потрібно використовувати для аналізу HTML-сторінок.
- Автоматизація завдань: Postman дозволяє автоматизувати завдання, такі як надсилання запитів і витяг даних. Це може бути корисно для економії часу та зусиль при аналізі великої кількості HTML-сторінок.

Postman має безкоштовну та платну версії.

Безкоштовна версія пропонує широкий спектр функцій, які підходять для більшості користувачів. Платна версія пропонує додаткові функції, такі як співпраця в команді та розширене тестування.

Окрім Postman, для аналізу HTML-сторінок можна використовувати й інші інструменти, такі як:

- Веб-браузери з інструментами розробника: Більшість веб-браузерів мають вбудовані інструменти розробника, які можна використовувати для перегляду HTML-коду веб-сторінок.

- Бібліотеки парсингу HTML: Існує багато бібліотек парсингу HTML, які можна використовувати для програматичного аналізу HTML-коду.

- Спеціалізовані інструменти аналізу HTML: Існує також кілька спеціалізованих інструментів аналізу HTML, які пропонують додаткові функції, такі як виявлення помилок і оптимізація коду.

Після отримання HTML сторінки, потрібно знайти елементи та їхні унікальні назви для того щоб за допомогою Jsoup знайти їх та отримати доступ до значень. Jsoup пропонує багате та гнучке API для отримання даних з отриманої сторінки. Ми можемо використовувати методи для пошуку елементів по класу, по атрибутах, по значенням атрибутів, по значенням атрибутів які починаються або мають в собі певні значення.

Зі сторінки з всіма вакансіями нам потрібно отримати всі елементи які мають атрибут id та його значення яке починається з job-item для подальшого парсингу в рекорди Vacancy.

Кожен елемент job-item обробляється окремо та з нього дістається вся нам потрібна інформація як – час публікації, назва компанії, назва вакансії, посилання на вакансію, коротка інформація (рівень англійської, дистанційна робота, чи вакансія на розробку особистого продукту, вакансія для розробки ЗСУ і т.п.), опис вакансії. Для пришвидшення цього процесу я вирішив зробити обробку у декількох потоках.

2.3 Реалізація функціоналу генерації супровідного листа

Для генерації супровідних листів я використовую OpenAI API та бібліотеку Spring AI яка дає спрощений доступ для роботи з широким спектром функцій OpenAI.

Spring AI – це проект з відкритим кодом, що надає бібліотеки Java для професійної інтеграції OpenAI API у Java-додатки. Він полегшує розробникам Java використання потужних можливостей OpenAI, таких як генерація тексту, переклад мов та створення різноманітного креативного контенту. Переваги Spring AI:

- Інтуїтивний API: Spring AI пропонує чіткі та зрозумілі API для роботи з широким спектром функцій OpenAI, що робить його зручним для розробників з будь-яким рівнем досвіду.

- Автоматизація завдань: Spring AI автоматизує рутинні завдання, пов'язані з OpenAI API, такі як аутентифікація, обробка запитів та інтерпретація відповідей, економлячи час та зусилля розробників.

- Зручність використання: Завдяки своїй простоті та інтуїтивності Spring AI робить OpenAI API доступним для ширшого кола розробників Java, навіть без попереднього досвіду роботи з API.

Для генерації супровідних листів на базі вакансії та інформації про користувача я створив спеціальний інтерфейс який потрібно реалізувати для роботи.

```
1 public interface CoverLetterService {
2 VacancyCoverLetter generateVacancyCoverLetter(BojiaBotUser user, Vacancy vacancy);
3 }
```

Перша його реалізація буде використовувати OpenAI ChatGPT 3.5 Turbo 0125 для генерації супровідних листів. У майбутньому можна буде без зайвих

проблем створити сервіс який наприклад буде взаємодіяти з іншим Gen-AI для тестування якості згенерованих супровідних листів. Найважливіша річ для генерації будь якого тексту, це написання та підбір гарного **prompt** (підказка) для отримання бажаного результату генерації.

Prompt, або підказка, – це інструкція або опис, що надається штучному інтелекту (ШІ) для генерування тексту[11].

Він слугує путівником для ШІ, даючи йому зрозуміти, який тип тексту потрібно створити, та надаючи контекст для його створення.

Prompt може бути простим реченням або складним описом, що містить деталі про стиль, тон, тему та інші аспекти тексту.

Ось кілька прикладів prompts:

- "Напишіть вірш про кохання."
- "Створіть новинну статтю про останні наукові відкриття."
- "Напишіть сценарій для короткометражного фільму про групу друзів, які вирушають у пригоди."
- "Створіть маркетинговий текст для нового продукту."
- "Напишіть електронний лист клієнту, щоб повідомити про затримку доставки."

Якість prompt значно впливає на якість тексту, який генерує ШІ.

Чим чіткіший, детальніший і інформативніший prompt, тим кращим буде результат.

При тестуванні застосунку було спробувана та протестована велика кількість prompt для генерації супровідного листа. Основні проблеми з якими я стикнувся при розробці це погано підкреслені рамки з якими повинен працювати ШІ. Щоб отримати гарно написаний супровідний лист я просив ШІ писати його як HR спеціаліст з великим досвідом роботи і який бачив багато гарних листів. Через

це прохання, результати іноді були не очікуваними, через що лист виглядав неначе написаним від лиця HR який рекомендував співробітника. Часто супровідні листи виходили дуже довгими, через що деякі сервіси навіть не дозволяли відправляти такі довгі тексти. Через певний час проб та помилок, оптимальним prompt-ом став:

```
1  "{description}".
2  Create a short cover letter (around 5 sentences) to this vacancy description above as a Senior HR specialist who saw a lot of nice cover letters. To generate a cover letter, use the information below, provided by a person interested in a vacancy: "{prompt}".
3  Remember, you are writing this cover letter for another person (not as a HR) with information about them above from their name. This cover letter will be immediately send from a person interested in a vacancy. You don't have to write a recommendation, just a cover letter example for interested person.
```

Слова які знаходяться у фігурних дужках, це інформація яка буде змінюватись для кожної вакансії та користувача для отримання унікальних результатів. Завдяки зручному інструменту від Spring AI під назвою PromptTemplate ми можемо за допомогою інтуітивного API змінювати написаний prompt окремо для кожної вакансії.

Під час тестування телеграм бота було отримано багато вакансій та згенеровано супровідних листів за допомогою prompt вище. Ось приклад одного зі згенерованих супровідних листів:

```
1  Dear Hiring Manager,
2  I am a Java Software Engineer with 4+ years of experience, proficient in Java 17, SpringBoot 3, CI/CD, Kafka, and AWS. I am excited about the opportunity to join your team at ThingsBoard Inc. and contribute to the development of your open-source IoT platform. My hands-on experience and technical skills align well with the requirements
```

for the Middle Java developer position. I look forward to the possibility of discussing how my background, skills, and enthusiasm can benefit your team.

3 Best regards, [Name]

Як видно, отриманий супровідний лист згенерований та персоналізований виключно під написану інформацію про користувача бота та опис вакансії. У ньому згадана інформація про досвід, технології, фреймворки та інструменти з якими працював користувач. Згадана назва компанії та позиція на яку ми хочемо податися. Також залишені місця де нам потрібно підставити ім’я. У майбутньому при бажанні можна додати функціонал до боту який просити ім’я користувача для автоматичного генерування супровідного листа відразу з ім’ям.

2.4 Збір інформації про кандидата

Збір інформації про кандидата є фундаментом для створення релевантного та персоналізованого супровідного листа, який збільшить шанси кандидата на успішне працевлаштування.

Чому це важливо?

- Персоналізація: Супровідний лист, адаптований до досвіду, навичок та інтересів кандидата, викликає більший інтерес у роботодавця, ніж шаблонний лист.

- Підкреслення відповідності: Збір інформації про кандидата дозволяє ідентифікувати ключові слова та фрази з опису вакансії, які можна включити до супровідного листа, щоб продемонструвати відповідність кандидата цій посаді.

- Свідчення про зацікавленість: Ретельний збір інформації про кандидата підкреслює його серйозність та зацікавленість у цій вакансії.

- Підвищення шансів на співбесіду: Персоналізований та релевантний супровідний лист може суттєво збільшити шанси кандидата на отримання запрошення на співбесіду.

Яку інформацію слід збирати?

- Професійний досвід: Попередні місця роботи, посади, дати роботи, ключові досягнення та відповідальності.
- Навички: Технічні, м'які, мовні навички.
- Інтереси: Професійні та особисті інтереси, які стосуються вакансії.
- Досягнення: Нагороди, сертифікати, публікації, інші досягнення, які можуть бути релевантними для вакансії.

Для збору інформації про кандидата, він же користувач боту, ми використаємо можливість бота реагувати на команди типу /prompt. Перед тим як додати будь який пошук, користувачу необхідно додати якусь інформацію про себе обравши з меню команду /prompt. Він може написати будь який текст про себе та свій досвід до 1024 знаків, згадавши про свої навички, роки досвіду, інструменти з якими він працював, бажання по роботі, релокації, технологіям і т.д. Саме якість цієї інформації буде впливати на генерацію супровідного листа. Чим точніше користувач напише про себе, ти точніше та краще буде лист згенерований ШІ для вакансії.

3 Експлуатація та оцінка телеграм-бота

3.1 Реагування телеграм бота на оновлення

Для того щоб отримувати повідомлення від користувача та розуміти його команди, телеграм бот повинен підписатися та реагувати на оновлення які ми отримуємо коли користувач взаємодіє з ботом.

Оновлення може мати у собі дуже великий спектр інформації який залежить від типу повідомлення яке було відправлене користувачем. Наш телеграм бот реагує тільки на оновлення з повідомленням або з callback інформацією яка надсилається після натисканя кнопоки. Для бота доступний такий список команд:

команда допомоги: ця команда розповідає базову інформацію про бота, його фукнціонал, як його використовувати, та список наявних команд

команда для додавання prompt: завдяки цій команді користувач може додати інформацію про себе та свій досвід яка повинна бути висвітлена у супровідному листі

команда пошуку: після виклику цієї команди ви можете ознайомитись зі своїми доданими пошуковими запитами та при бажані видалити той чи інший пошук зі системи

команда видалення пошуку: передавши до цієї команди номер пошуку, він буде видалений зі списку пошуків і ви більше не будете отримувати сповіжень та супровідних листів коли на сайті з’являється нова вакансія яка підходить під цей пошуковий запит

команда djinni: виконавши цю команду ви можете додати ключові слова для пошуку вакансії на сайті Djinni, бот буде з певним інтервалом перевіряти

наявність нових вакансій та відправляти вам інформацію про нові вакансії разом зі згенерованим супровідним листом

Крім команд, бот також реагує на callback інформацію яка відправляється коли користувач натискає на кнопку яка прикріплена до певного повідомлення. Зараз кнопки наявні тільки для команд які додають пошук до системи. Коли користувач виконує команду /djinni middle java бот надсилає повідомлення яке має інформацію про наявну кількість вакансій які підпадають під цей пошуковий запит та запитує чи ви хочете зберегти цей пошук для отримування нотифікацій коли щось нове з’являється на сайті. При натисканні кнопки «Так» ваш пошук з ключовими словами залишиться у базі, та буде брати участь при тримані нових вакансій, після того як ви натисните кнопку ні цей пошук буде видалений, та ви не будете отримувати сповіщень про нові вакансії та згенеровані супровідні листи.

3.2 Batch Job для отримання вакансій та генерації супровідних листів

Розробик або користувач який запускає бота може без будь-яких проблем задати інтервал з яким потрібно перевіряти наявність нових вакансій та генерації для них супровідних листів. Кількість пошукових запитів може бути дуже великою, тому для оброки такого набору даних була використана бібліотека Spring Batch яка надає доступ до створення перевикористовуваних функцій які є суттєвими у питанні обробки великої кількості даних, включаючи логування та нагляд виконання, менеджмент транзакцій, статистика виконання роботи, рестар роботи, пропуск виконання та керування ресурсами.

```
1    @Scheduled(cron = "${bojia.cover-letter-job-cron}")
2        public void execute() {
3            try {
4                this.jobLauncher.run(this.coverLettersJob, new JobParameters());
5            } catch (Exception ex) {
6                this.exceptionHandlerService.publishException(ex);
```

```
7               }
8           }
```

Зі заданим інтервалом сервер на якому знаходиться телеграм бот, буде за одну ітерацію брати 100 пошукових запитів з бази під час етапу читання інформації, в наступному етапі для кожного з них отримувати останні опубліковані вакансії, та для кожної вакансії генерувати супровідний лист для користувача, і в останьому етапі відправляти вакансію та супровідний лист користувачу.

Кожна ітерація та етап виконуються в окремому потоці, через що IO запити будуть виконуватись швидше та робота буде виконуватись з використанням меншої кількості ресурсів.

Завдяки гарно створеному API Spring Batch опис та створення роботи дуже легкий та зрозумілий. Нам потрібно визначити всього 3 елементи – IterReader який буде читати дані з бази даних, IterProcessor який буде обробляти та перетворювати (при бажані) отримані дані в іншу структуру та IterWriter який буде записувати отриманні значення до бази даних або відправляти до іншої системи. Після чого зібрати їх у Step та визначити Job яка буде виконуватися:

```
1           @Bean
2           Job coverLettersJob(
3                   JobRepository jobRepository,
4                   Step step1
5           ) {
6               return new JobBuilder("coverLettersJob", jobRepository)
7                       .incrementer(new RunIdIncrementer())
8                       .start(step1)
9                       .build();
10          }
```

```
11
12      @Bean
13      Step step1(
14              JobRepository jobRepository,
15              PlatformTransactionManager platformTransactionManager,
16              JpaPagingItemReader<BojiaBotUserSearch> botUserSearchReader,
17              BojiaBotUserSearchProcessor botUserSearchProcessor,
18              VacanciesCoverLettersWriter vacanciesCoverLettersWriter
19      ) {
20          return new StepBuilder("step1", jobRepository)
21                  .<BojiaBotUserSearch, VacanciesCoverLetters>chunk(10,
platformTransactionManager)
22                  .reader(botUserSearchReader)
23                  .processor(botUserSearchProcessor)
24                  .writer(vacanciesCoverLettersWriter)
25                  .allowStartIfComplete(true)
26                  .taskExecutor(new SimpleAsyncTaskExecutor("coverLettersJob-step1-
"))
27                  .build();
28      }
```

3.3 Оцінка роботи бота та якості згенерованих листів

Запуск бота відбувається за допомою скрипта який знаходиться у корені репозиторія та називається run.sh. Для запуску бота вам потрібно за допомогою іншого бота з назвою BotFather зареєструвати свого бота та отримати токен який потрібно вставити у TELEGRAM_BOT_TOKEN замість token-text. Після цього вам потрібно зареєструватися на сайті OpenAI, отримати ключ та вставити його у OPENAI_API_KEY замість api-key. Для зберігання всіх даних вам потрібно встановити PostgreSQL та створити у ньому базу з назвою наприклад bojia-tg-bot

та налаштувати у скрипті значення DATABASE_URL, DATABASE_USERNAME, DATABASE_PASSWORD.

```
1  #!/usr/bin/env sh
2
3  PROFILE="dev"
4  DATABASE_NAME="bojia-tg-bot"
5  DATABASE_URL="jdbc:postgresql://localhost:5432/$DATABASE_NAME"
6  DATABASE_USERNAME="postgres"
7  DATABASE_PASSWORD="postgres"
8  TELEGRAM_BOT_TOKEN="token-text"
9  OPENAI_API_KEY="api-key"
10
11 set -- \
12   --spring.profiles.active="$PROFILE" \
13   --spring.datasource.url="$DATABASE_URL" \
14   --spring.datasource.username="$DATABASE_USERNAME" \
15   --spring.datasource.password="$DATABASE_PASSWORD" \
16   --bojia.bot-token="$TELEGRAM_BOT_TOKEN" \
17   --spring.ai.openai.api-key="$OPENAI_API_KEY"
18
19 echo "./mvnw spring-boot:run -Dspring-boot.run.arguments=\"$*\""
20
21 ./mvnw spring-boot:run -Dspring-boot.run.arguments="$*"
```

Для правильного функціоналу застусунку ви маєте встановити Java 17 у path та запустити цей скрипт який збудує вихідний код та запустить бота.

За допомогою команд бота /prompt та /djinni я додав таку інформацію про себе:

- I'm Java Software Engineer with 4+ years of experience. Worked with Java 17, SpringBoot 3, CI/CD, Kafka, AWS.

Та додав такі пошукові запити для пошуку вакансій на сайті Djinni:

- senior java
- middle java
- junior java
- java

Через певний час я почав отримувати повідомлення про нові ваканції у такому форматі

1 **Senior Java Developer** at FxPro
2 *Релокейт, Кіпр · Продукт · Гібридна робота · 5 років досвіду · Intermediate*
3 Join FxPro: a leading international fintech company. Be a part of our expanding international team, with offices in Limassol, London, Monaco, Nassau, and Dubai. FxPro boasts a
4 Join FxPro: a leading international fintech company. Be a part of our expandi...
5 https://djinni.co/jobs/url-deleted
6 *Subscription:* senior java *(Djinni)*

Та відразу відповіддю на це повідомлення був згенерований супровідний лист відповідно до опису вакансії та тієї інформації яку я надав про себе боту раніше. Завдяки чому є можливіть без зайвого очікування скопіювати текст, додати своє ім’я та відправити відповідь на вакансію разом зі своїм резюме. Повідомлення зі згенерованим супровідним листом виглядає ось так:

1 Generated cover letter for this vacancy:
2
3 Dear Hiring Manager,
4

I am a Java Software Engineer with over 4 years of experience, specializing in Java 17, SpringBoot 3, CI/CD, Kafka, and AWS. I have a proven track record of designing, developing, and maintaining Java-based backend applications, collaborating effectively with cross-functional teams, and optimizing application performance. I am passionate about coding, problem-solving, and staying up-to-date with emerging technologies. I am excited about the opportunity to contribute to your dynamic start-up project in digital marketing and make a significant impact on the team.

Thank you for considering my application.

Sincerely,
[Your Name]

Як можна побачити, супровідний лист згенерований ідеально та містить інформацію яку ми надали про себе разом з інформацією з вакансії, що може додати вам плюс як кандидату після швидкої відповіді на вакансію, так як ви будеше у спису перших відповідей з гарним супровідним листом який зацікавить кожного рекрутера.

ВИСНОВКИ

У цьому дослідженні представлено розробку телеграм-бота на мові програмування Java, призначеного для автоматизованого пошуку вакансій та генерації супровідних листів[1].

Розроблений телеграм-бот володіє наступними функціональними можливостями:

- Пошук вакансій: Забезпечує пошук вакансій за ключовими словами на різних сайтах провайдерах вакансій.
- Збереження вакансій: Надає можливість зберігати знайдені вакансії для подальшого перегляду та аналізу.
- Генерація супровідного листа: Автоматично генерує супровідний лист на основі опису кандидата та опису вакансії.

Проведено тестування бота на реальних даних, яке підтвердило його високу ефективність.

Бот демонструє здатність знаходити релевантні вакансії, генерувати якісні супровідні листи та економити час користувачам.

Практична цінність розробленого телеграм-бота полягає в:

- Економії часу: Бот значно економить час користувачів, автоматизуючи процес пошуку вакансій та написання супровідних листів.
- Підвищенні шансів на працевлаштування: Допомагає користувачам знаходити релевантні вакансії та генерувати супровідні листи, які роблять їх більш конкурентними на ринку праці.
- Доступності: Завдяки доступності через Telegram, бот має широкий спектр потенційних користувачів.

Результати дипломної роботи демонструють, що телеграм-боти можуть бути ефективним інструментом для автоматизації завдань, пов'язаних з пошуком роботи.

Розроблені алгоритми та методи можуть бути використані для створення інших ботів, що допомагають користувачам у різних сферах життя.

Перспективні напрямки досліджень у цій галузі:

- Розробка більш складних алгоритмів пошуку та фільтрації вакансій.
- Вдосконалення методів генерації супровідних листів.
- Інтеграція бота з іншими системами та сервісами.
- Дослідження етичних аспектів використання ботів для пошуку роботи.

Загалом, розробка телеграм-бота для автоматизованого пошуку вакансій і генерації супровідного листа є актуальним та перспективним завданням, яке має значний потенціал для практичного застосування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Bojia Telegram Bot [Електроний ресурс] – режим доступу: https://github.com/volodbol/bojia-tg-bot
2. Bots: An introduction for developers [Електронний ресурс] – режим доступу: https://core.telegram.org/bots
3. Dialogflow [Електронний ресурс] – режим доступу: https://cloud.google.com/dialogflow/es/docs/integrations/telegram
4. FlowXO [Електронний ресурс] – режим доступу: https://flowxo.com/channels/telegram/
5. Jooble | Робочі вакансії [Електронний ресурс] – режим доступу: https://t.me/Jooble_jobs_bot
6. Manybot [Електронний ресурс] – режим доступу: https://manybot.io/
7. Number of monthly active Telegram users worldwide from March 2014 to July 2023 [Електронний ресурс] – режим доступу: https://www.statista.com/statistics/234038/telegram-messenger-mau-users/
8. Postman API platform [Електронний ресурс] – режим доступу: https://www.postman.com/
9. Rasa Telegram Connector [Електронний ресурс] – режим доступу: https://rasa.com/docs/rasa/connectors/telegram/
10. robota.ua NOW Вакансії [Електронний ресурс] – режим доступу: https://t.me/robotaua_now_bot
11. Spring AI Reference AI Concepts Prompts [Електронний ресурс] – режим доступу: https://docs.spring.io/spring-ai/reference/concepts.html#_prompts
12. Telegram Bot API [Електроний ресурс] – режим доступу: https://core.telegram.org/bots/api
13. Telegram Bot Building: A Guide to Building Your ChatGPT Telegram Bot [Електронний ресурс] – режим доступу: https://yourgpt.ai/blog/growth/telegram-bot-building-a-guide-to-building-your-chatgpt-telegram-bot

14. Work.ua Нові вакансії [Електронний ресурс] – режим доступу: https://t.me/jobs_workua_bot
15. Вакансії на Джині [Електронний ресурс] – режим доступу: https://t.me/djinni_jobs_bot

ДОДАТОК А

КОД ПРОГРАМИ

```
package com.volod.bojia.tg.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.retry.backoff.ExponentialBackOffPolicy;
import org.springframework.retry.policy.SimpleRetryPolicy;
import org.springframework.retry.support.RetryTemplate;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.scheduling.annotation.EnableScheduling;

import java.time.Clock;
import java.time.ZoneId;
import java.util.concurrent.TimeUnit;

@EnableScheduling
@EnableAsync
@Configuration
public class BojiaApplicationConfiguration {

    @Bean
    public RetryTemplate retryTemplate() {
        var retryTemplate = new RetryTemplate(
```

```
        var exponentialBackOffPolicy = new ExponentialBackOffPolicy();
        exponentialBackOffPolicy.setInitialInterval(TimeUnit.SECONDS.toMillis(2));
        exponentialBackOffPolicy.setMultiplier(5);
        exponentialBackOffPolicy.setMaxInterval(TimeUnit.MINUTES.toMillis(1));
        retryTemplate.setBackOffPolicy(exponentialBackOffPolicy);

        var simpleRetryPolicy = new SimpleRetryPolicy();
        simpleRetryPolicy.setMaxAttempts(3);
        retryTemplate.setRetryPolicy(simpleRetryPolicy);

        return retryTemplate;
    }

    @Bean
    public Clock clock() {
        return Clock.system(ZoneId.of("Europe/Kyiv"));
    }

}
```

```
package com.volod.bojia.tg.configuration;

import com.pengrad.telegrambot.TelegramBot;
import com.pengrad.telegrambot.request.GetMe;
import com.volod.bojia.tg.bot.handler.BojiaBotExceptionHandler;
import com.volod.bojia.tg.bot.listener.BojiaBotUpdatesListener;
```

```
import com.volod.bojia.tg.constant.BojiaLogConstants;
import com.volod.bojia.tg.domain.exception.BojiaBotInitializationException;
import com.volod.bojia.tg.property.BojiaApplicationProperties;
import com.volod.bojia.tg.service.bot.BojiaBotUpdateService;
import com.volod.bojia.tg.service.exception.BojiaExceptionHandlerService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;

import java.util.List;

@Slf4j
@Profile("dev")
@Configuration
@RequiredArgsConstructor
public class BojiaBotConfiguration {

    private final BojiaApplicationProperties applicationProperties;

    @Bean
    public TelegramBot bojiaBot() throws BojiaBotInitializationException {
        var bot = new TelegramBot(this.applicationProperties.getBotToken());
        try {
```

```
            var getMeResponse = bot.execute(new GetMe());
            LOGGER.trace(BojiaLogConstants.BOT_PREFIX + "getMeResponse: {}", getMeResponse);
            if (!getMeResponse.isOk()) {
                throw new IllegalArgumentException("Token is invalid. GetMe response - [%s]".formatted(getMeResponse));
            }
        } catch (RuntimeException ex) {
            throw new BojiaBotInitializationException("Can't initialize bojia bot", ex);
        }
        return bot;
    }

    @Bean
    public BojiaBotUpdatesListener updatesListener(
            List<BojiaBotUpdateService> updateServices,
            BojiaExceptionHandlerService exceptionHandlerService
    ) {
        return new BojiaBotUpdatesListener(
                updateServices,
                exceptionHandlerService
        );
    }

    @Bean
    public BojiaBotExceptionHandler exceptionHandler(
```

```
            BojiaExceptionHandlerService bojiaExceptionHandlerService
    ) {
        return new BojiaBotExceptionHandler(
                bojiaExceptionHandlerService
        );
    }

}
```

```
package com.volod.bojia.tg.configuration;

import
org.springframework.beans.factory.support.BeanDefinitionRegistryPostProcessor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class BojiaSpringBatchConfiguration {

    // https://github.com/spring-projects/spring-batch/issues/4519#issuecomment-
1895372351
    // https://github.com/spring-projects/spring-batch/issues/4519#issuecomment-
1860530016
    @Bean
    public static BeanDefinitionRegistryPostProcessor
jobRegistryBeanPostProcessorRemover() {
```

```
        return registry ->
registry.removeBeanDefinition("jobRegistryBeanPostProcessor");
    }

}
```

```
package com.volod.bojia.tg.bot.listener;

import com.pengrad.telegrambot.UpdatesListener;
import com.pengrad.telegrambot.model.Update;
import com.volod.bojia.tg.service.bot.BojiaBotUpdateService;
import com.volod.bojia.tg.service.exception.BojiaExceptionHandlerService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;

import java.util.List;

@Slf4j
@RequiredArgsConstructor
public class BojiaBotUpdatesListener implements UpdatesListener {

    private final List<BojiaBotUpdateService> updateServices;
    private final BojiaExceptionHandlerService exceptionHandlerService;

    @Override
```

```
    public int process(List<Update> updates) {
        try {
            LOGGER.trace("Updates - {}", updates);
            return updates.stream()
                    .map(update -> this.updateServices.stream()
                            .filter(service -> service.isUpdateValid(update))
                            .findFirst()
                            .map(service -> service.processUpdate(update))
                            .orElse(CONFIRMED_UPDATES_ALL)
                    )
                    .max(Integer::compareTo)
                    .orElse(CONFIRMED_UPDATES_ALL);
        } catch (RuntimeException ex) {
            this.exceptionHandlerService.publishException(ex);
            return CONFIRMED_UPDATES_ALL;
        }
    }

}

package com.volod.bojia.tg.bot.handler;

import com.pengrad.telegrambot.ExceptionHandler;
import com.pengrad.telegrambot.TelegramException;
import com.volod.bojia.tg.service.exception.BojiaExceptionHandlerService;
```

```
import lombok.RequiredArgsConstructor;

@RequiredArgsConstructor
public class BojiaBotExceptionHandler implements ExceptionHandler {

    private final BojiaExceptionHandlerService exceptionHandlerService;

    @Override
    public void onException(TelegramException ex) {
        this.exceptionHandlerService.publishException(ex);
    }

}

package com.volod.bojia.tg.constant;

import lombok.experimental.UtilityClass;

@UtilityClass
public class BojiaLogConstants {
    public final String APPLICATION_PREFIX = "[Application] ";
    public final String BOT_PREFIX = "[Bojia Bot] ";
    public final String BOT_ERROR = BOT_PREFIX + "error occurred. ";
}
```

```
package com.volod.bojia.tg.constant;

import lombok.experimental.UtilityClass;

@UtilityClass
public class JsoupConstants {
    public final String ID = "id";
    public final String CLASS = "class";
    public final String HREF = "href";
    public final String TITLE = "title";
}

package com.volod.bojia.tg.constant;

import lombok.experimental.UtilityClass;

@UtilityClass
public class PostgresConstants {
    public final String BOJIA_BOT_DETAILS_TABLE = "bojia_bot_details";
    public final String BOJIA_BOT_USER_TABLE = "bojia_bot_user";
    public final String BOJIA_BOT_USER_SEARCH = "bojia_bot_user_search";
}

package com.volod.bojia.tg.cron;

import com.volod.bojia.tg.service.exception.BojiaExceptionHandlerService;
```

```
import lombok.RequiredArgsConstructor;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class CoverLetterJobCron {

    private final JobLauncher jobLauncher;
    private final Job coverLettersJob;
    private final BojiaExceptionHandlerService exceptionHandlerService;

    @Scheduled(cron = "${bojia.cover-letter-job-cron}")
    public void execute() {
        try {
            this.jobLauncher.run(this.coverLettersJob, new JobParameters());
        } catch (Exception ex) {
            this.exceptionHandlerService.publishException(ex);
        }
    }

}
```

```
package com.volod.bojia.tg.domain.bot;

import lombok.AllArgsConstructor;
import lombok.Getter;

@AllArgsConstructor
@Getter
public enum BojiaBotCallback {
    SEARCH_SAVED("ss"),
    SEARCH_REMOVED("sr");

    private final String value;

    public static String data(BojiaBotCallback callback, Long data) {
        return data(callback, data.toString());
    }

    public static String data(BojiaBotCallback callback, String data) {
        return "%s_%s".formatted(callback.value, data);
    }

    public int length() {
        return this.value.length() + 1;
    }
}
```

```
package com.volod.bojia.tg.domain.bot;

import com.pengrad.telegrambot.model.BotCommand;
import lombok.AllArgsConstructor;
import lombok.Getter;

import java.util.Arrays;
import java.util.stream.Collectors;

@AllArgsConstructor
@Getter
public enum BojiaBotCommand {
    HELP("help", "see help page"),
    ADD_PROMPT("prompt", "add prompt to use for cover letter generation"),
    SEARCHES("searches", "list of all searches"),
    REMOVE_SEARCH("remove", "remove search from use"),
    DJINNI("djinni", "add djinni search"),;

    private final String value;
    private final String description;

    public String getValue() {
        return "/" + this.value;
    }

    public static BotCommand[] getBotCommands() {
```

```
        return Arrays.stream(BojiaBotCommand.values())
                .map(command -> new BotCommand(command.getValue(),
command.getDescription()))
                .toArray(BotCommand[]::new);
    }

    public static String getJoinedCommands() {
        return Arrays.stream(BojiaBotCommand.values())
                .map(BojiaBotCommand::getValue)
                .collect(Collectors.joining("\n"));
    }

}
```

```
package com.volod.bojia.tg.domain.bot;

import com.pengrad.telegrambot.model.Update;
import com.pengrad.telegrambot.model.request.ParseMode;
import com.pengrad.telegrambot.request.EditMessageText;
import com.pengrad.telegrambot.request.SendMessage;
import lombok.*;

import java.util.regex.Pattern;

import static java.util.Objects.nonNull;
```

```
@Getter
@RequiredArgsConstructor(access = AccessLevel.PRIVATE)
@EqualsAndHashCode
@ToString
public class MessageMarkdownV2 {
    private final Object chatId;
    private final Integer messageId;
    private final String value;

    public static MessageMarkdownV2Builder builder() {
        return new MessageMarkdownV2Builder();
    }

    public SendMessage toSendMessage() {
        return new SendMessage(this.chatId,
this.value).parseMode(ParseMode.MarkdownV2);
    }

    public EditMessageText toEditMessageText() {
        return new EditMessageText(this.chatId, this.messageId,
this.value).parseMode(ParseMode.MarkdownV2);
    }

    public static class MessageMarkdownV2Builder {
```

```
    private static final Pattern ESCAPE_PATTERN =
Pattern.compile("([_*\\[\\]()~`>#+\\-=|{}.!])");

    private Object chatId;
    private Integer messageId;
    private final StringBuilder builder;

    private MessageMarkdownV2Builder() {
        this.builder = new StringBuilder();
    }

    private String escape(String value) {
        return ESCAPE_PATTERN.matcher(value).replaceAll("\\\\$1");
    }

    public MessageMarkdownV2Builder chatId(Object chatId) {
        this.chatId = chatId;
        return this;
    }

    public MessageMarkdownV2Builder chatId(Update update) {
        if (nonNull(update.message())) {
            return this.chatId(update.message().chat().id());
        } else if (nonNull(update.callbackQuery())) {
            return
this.chatId(update.callbackQuery().maybeInaccessibleMessage().chat().id());
```

```
        } else {
            throw new IllegalArgumentException("ChatId is unavailable");
        }
    }

    public MessageMarkdownV2Builder messageId(Integer messageId) {
        this.messageId = messageId;
        return this;
    }

    public MessageMarkdownV2Builder messageId(Update update) {
        if (nonNull(update.message())) {
            return this.messageId(update.message().messageId());
        } else if (nonNull(update.callbackQuery())) {
            return
this.messageId(update.callbackQuery().maybeInaccessibleMessage().messageId());
        } else {
            throw new IllegalArgumentException("MessageId is unavailable");
        }
    }

    public MessageMarkdownV2Builder text(String value) {
        this.builder.append(this.escape(value));
        return this;
    }
```

```
public MessageMarkdownV2Builder inlineCode(String value) {
    this.builder.append("`%s`".formatted(this.escape(value)));
    return this;
}

public MessageMarkdownV2Builder bold(String value) {
    this.builder.append("*%s*".formatted(this.escape(value)));
    return this;
}

public MessageMarkdownV2Builder italic(String value) {
    this.builder.append("_%s_".formatted(this.escape(value)));
    return this;
}

public MessageMarkdownV2 build() {
    if (nonNull(this.chatId)) {
        var message = this.builder.toString();
        return new MessageMarkdownV2(
                this.chatId,
                this.messageId,
                message
        );
    } else {
```

```
                throw new IllegalArgumentException("ChatId is null");
            }
        }

    }
}
```

```
package com.volod.bojia.tg.domain.exception;

public class BojiaBotInitializationException extends Exception {

    public BojiaBotInitializationException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

```
package com.volod.bojia.tg.domain.exception;

import com.pengrad.telegrambot.model.Update;

public class BojiaBotUpdateIllegal extends Exception {

    public BojiaBotUpdateIllegal(Update update) {
        super("Illegal update: " + update);
    }
```

```
}

package com.volod.bojia.tg.domain.exception;

import com.volod.bojia.tg.domain.vacancy.VacancyProvider;
import org.jsoup.nodes.Element;

public class BojiaVacancyParseException extends Exception {

    public BojiaVacancyParseException(String message, VacancyProvider provider,
Throwable cause) {
        super(
                "%s. Provider: %s".formatted(message, provider),
                cause
        );
    }

    public static BojiaVacancyParseException getVacancies(VacancyProvider provider,
Throwable cause) {
        return new BojiaVacancyParseException("Can't get last vacancies", provider,
cause);
    }

    public static BojiaVacancyParseException getNumberOfVacancies(VacancyProvider
provider, Throwable cause) {
```

```
        return new BojiaVacancyParseException("Can't get number of vacancies",
provider, cause);
    }

    public static BojiaVacancyParseException parseError(Element element,
VacancyProvider provider, Throwable cause) {
        return new BojiaVacancyParseException("Can't parse vacancy.
%s".formatted(element), provider, cause);
    }

}
```

```
package com.volod.bojia.tg.domain.letter;

import com.volod.bojia.tg.entity.BojiaBotUserSearch;

import java.time.Instant;
import java.util.List;

public record VacanciesCoverLetters(
        BojiaBotUserSearch botUserSearch,
        List<VacancyCoverLetter> values,
        Instant lastPublishedVacancy
) {

    public static VacanciesCoverLetters of(
```

```
            BojiaBotUserSearch botUserSearch,
            List<VacancyCoverLetter> vacancyCoverLetters
    ) {
        return new VacanciesCoverLetters(
                botUserSearch,
                vacancyCoverLetters,
                vacancyCoverLetters.stream()
                        .map(letter -> letter.vacancy().published())
                        .max(Instant::compareTo)
                        .orElseGet(Instant::now)
        );
    }

}

package com.volod.bojia.tg.domain.letter;

import com.volod.bojia.tg.domain.vacancy.Vacancy;
import com.volod.bojia.tg.entity.BojiaBotUser;

public record VacancyCoverLetter(
        BojiaBotUser user,
        Vacancy vacancy,
        String value
) {
```

```
}

package com.volod.bojia.tg.domain.search;

import com.pengrad.telegrambot.model.Update;
import com.volod.bojia.tg.domain.vacancy.VacancyProvider;

import java.util.List;

import static java.util.Objects.isNull;

public abstract class AddSearchMiddleware {
    private AddSearchMiddleware next;

    public static AddSearchMiddleware link(AddSearchMiddleware first,
List<AddSearchMiddleware> chain) {
        var head = first;
        for (var nextInChain : chain) {
            head.next = nextInChain;
            head = nextInChain;
        }
        return first;
    }

    public abstract boolean check(Update update, VacancyProvider provider);
```

```
    protected boolean checkNext(Update update, VacancyProvider provider) {
        if (isNull(this.next)) {
            return true;
        }
        return this.next.check(update, provider);
    }
}

package com.volod.bojia.tg.domain.search;

import com.volod.bojia.tg.entity.BojiaBotUserSearch;

import java.util.List;
import java.util.stream.Collectors;

import static org.springframework.util.CollectionUtils.isEmpty;

public record BojiaBotUserSearches(List<BojiaBotUserSearch> values) {

    public String getKeywordsOrDefault() {
        if (isEmpty(this.values)) {
            return "You have no searches";
        } else {
            return this.values.stream()
```

```
                .map(search -> "%s - %s - %s".formatted(
                        search.getId(),
                        search.getProvider().getReadableName(),
                        search.getKeywords()
                ))
                .collect(Collectors.joining("\n"));
        }
    }

}
```

ДОДАТОК Б

ПРЕЗЕНТАЦІЯ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

РОЗРОБКА ТЕЛЕГРАМ -БОТА НА МОВІ ПРОГРАМУВАННЯ JAVA ДЛЯ АВТОМАТИЗОВАНОГО ПОШУКУ ВАКАНСІЙ І ГЕНЕРАЦІЇ СУПРОВІДНОГО ЛИСТА

Виконав: студент 4 курсу, групи КНД-41

Большаков В.Р.

Керівник: к. т. н., доцент каф. КН

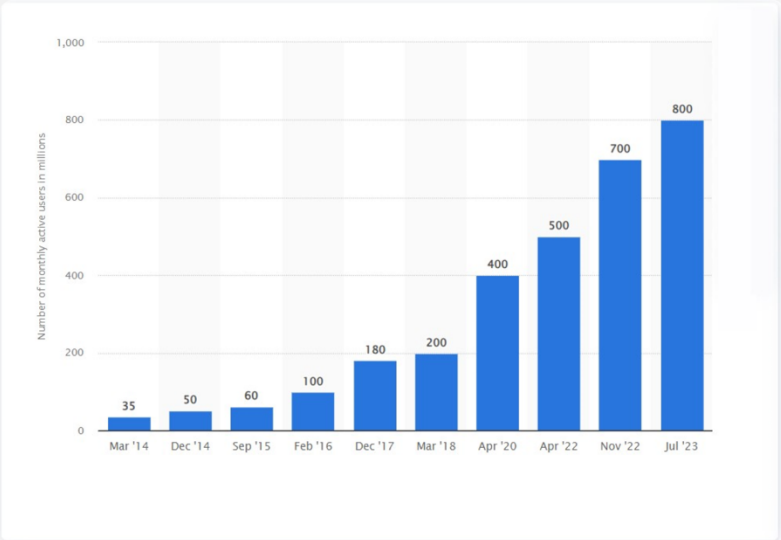
Іщеряков С.М.

м. Київ 2024р.

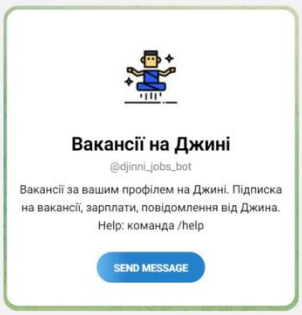
АКТУАЛЬНІСТЬ РОБОТИ

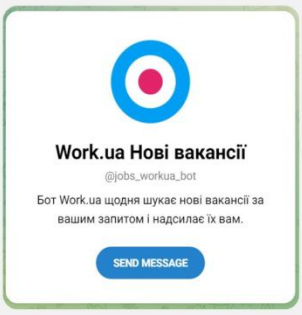
- *Об'єкт дослідження* – автоматизований пошук вакансій та генерація супровідного листа за допомогою телеграм-бота, розробленого на мові програмування Java.
- *Предмет роботи* – програма-телеграм-бот, що використовує Java для автоматизованого пошуку вакансій та породжувальний штучний інтелект для генерації супровідних листів.
- *Мета роботи* – розробка телеграм-бота, який допомагає користувачам знаходити релевантні вакансії та економити час на написанні супровідних листів.
- *Завдання роботи* – проаналізувати існуючі методи автоматизованого пошуку вакансій та генерації супровідних листів; розробити алгоритми для пошуку вакансій та генерації супровідних листів; створити телеграм-бота на мові програмування Java, який використовує розроблені алгоритми; провести тестування телеграм-бота та оцінити його ефективність.

КІЛЬКІСТЬ АКТИВНИХ КОРИСТУВАЧІВ TELEGRAM

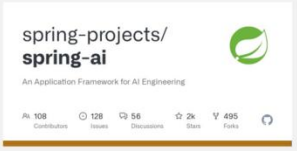


БОТИ ДЛЯ ПОШУКУ РОБОТ И

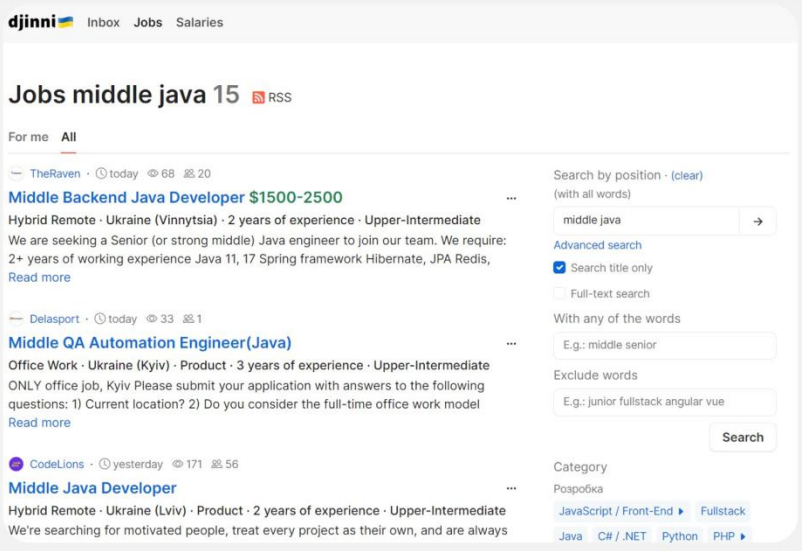




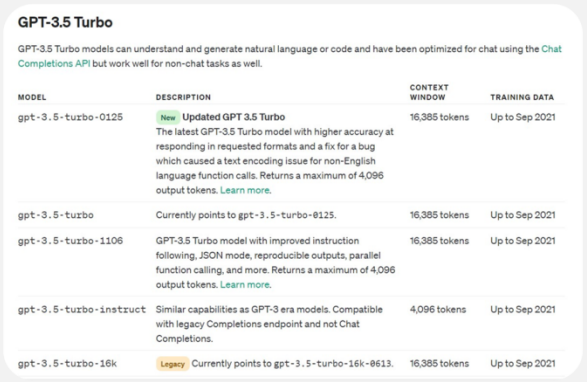
ІНТСРУМЕНТИ РОЗРОБК И



ПРОВАЙДЕР ВАКАНСІЙ



ПОРОДЖУВАЛЬНИЙ ШТУЧНИЙ ІНТЕЛЕК Т



GPT-3.5 Turbo

GPT-3.5 Turbo models can understand and generate natural language or code and have been optimized for chat using the Chat Completions API but work well for non-chat tasks as well.

MODEL	DESCRIPTION	CONTEXT WINDOW	TRAINING DATA
gpt-3.5-turbo-0125	New **Updated GPT 3.5 Turbo** The latest GPT-3.5 Turbo model with higher accuracy at responding in requested formats and a fix for a bug which caused a text encoding issue for non-English language function calls. Returns a maximum of 4,096 output tokens. Learn more.	16,385 tokens	Up to Sep 2021
gpt-3.5-turbo	Currently points to gpt-3.5-turbo-0125.	16,385 tokens	Up to Sep 2021
gpt-3.5-turbo-1106	GPT-3.5 Turbo model with improved instruction following, JSON mode, reproducible outputs, parallel function calling, and more. Returns a maximum of 4,096 output tokens. Learn more.	16,385 tokens	Up to Sep 2021
gpt-3.5-turbo-instruct	Similar capabilities as GPT-3 era models. Compatible with legacy Completions endpoint and not Chat Completions.	4,096 tokens	Up to Sep 2021
gpt-3.5-turbo-16k	Legacy Currently points to gpt-3.5-turbo-16k-0613.	16,385 tokens	Up to Sep 2021

ЕКСПЛУАТАЦІЯ БОТА

ЕКСПЛУАТАЦІЯ БОТА

ВИСНОВКИ

У цьому дослідженні представлено розробку телеграм -бота на мові програмування Java, призначеного для автоматизованого пошуку вакансій та генерації супровідних листів.

Розроблений телеграм-бот володіє наступними функціональними можливостями:

- Пошук вакансій: Забезпечує пошук вакансій за ключовими словами на різних сайтах провайдерах вакансій.
- Збереження вакансій: Надає можливість зберігати знайдені вакансії для подальшого перегляду та аналізу.
- Генерація супровідного листа: Автоматично генерує супровідний лист на основі опису кандидата та опису вакансії.

Проведено тестування бота на реальних даних, яке підтвердило його високу ефективність.

Бот демонструє здатність знаходити релевантні вакансії, генерувати якісні супровідні листи та економити час користувачам.

АПРОБАЦІЯ

За матеріалами були опубліковані тези:

- Большаков В.Р., Іщеряков С.М.,Автоматизація генерації супровідних листів для швидшого реагування на вакансії // СУЧАСНІ ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В НАУЦІ ТА ОСВІТІ: Матеріали міжнародної науково-практичної конференції. Збірник тез. 15.05.2024, ДУІКТ, м.Київ –К.: ДУІКТ, 2024.– с. 2.
- Большаков В.Р., Іщеряков С.М.,Етичні аспекти застосування штучного інтелекту в науковій та освітній діяльності // СУЧАСНІ ІНТЕЛЕКТУАЛЬНІ ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В НАУЦІ ТА ОСВІТІ: Матеріали міжнародної науково-практичної конференції. Збірник тез. 15.05.2024, ДУІКТ, м. Київ –К.: ДУІКТ, 2024.– с. 2.