Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

факультет прикладних інформаційних технологій та електроінженерії
(повна назва факультету)
кафедра автоматизації технологічних процесів і виробництв
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: «Розроблення персонального AI-асистента на основі великих мовних моделей з інтеграцією Google Workspace та автоматизацією бізнес-процесів»

Виконав(ла): студент(ка) IV курсу, групи КАс-41
спеціальності 174«Автоматизація, комп'ютерно-інтегровані технології та робототехніка
(шифр і назва спеціальності)

	(підпис)	Кіндій Д.Р. (прізвище та ініціали)
	(підпис)	(прізвище та ініціали)
Керівник	(підпис)	Коноваленко І.В. (прізвище та ініціали)
Нормоконтроль	(підпис)	Козбур І.Р. (прізвище та ініціали)
Завідувач кафедри	(підпис)	Савків В.Б. (прізвище та ініціали)
Рецензент	(підпис)	Стухляк Д.П. (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет прикладних інформаційних технологій та електроінженерії
(повна назва факультету)
Кафедра автоматизації технологічних процесів і виробництв
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Савків В.Б.
(підпис) (прізвище та ініціали)
« ___» _____2026р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)
за спеціальністю 151 «Автоматизація та комп'ютерно-інтегровані технології»
(шифр і назва спеціальності)
студенту Кіндій Денис Романович
(прізвище, ім'я, по батькові)

1. Тема роботи «Розроблення персонального AI-асистента на основі великих мовних моделей з інтеграцією Google Workspace та автоматизацією бізнес-процесів».

Керівник роботи Коноваленко І.В.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджена наказом ректора від « 14 » травня 2026 року № 4/9-231
2. Термін подання студентом завершеної роботи 26 червня 2026 року
3. Вихідні дані до роботи Програмний проєкт Personal Job Helper, вимоги до персонального AI-асистента для пошуку вакансій, обробки CV та AI-матчингу.
Технології Django, PostgreSQL, pgvector, OpenAI API, LangChain, Docker.
Інтеграція Gmail SMTP для нотифікацій про релевантні вакансії.
4. Зміст роботи (перелік питань, які потрібно розробити)
1) Аналітичний огляд предметної області та існуючих рішень;
2) Проєктування архітектури персонального AI-асистента;
3) Реалізація, розгортання та експлуатація програмної системи;
4) Економічне обґрунтування, безпека життєдіяльності та охорона праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Презентація кваліфікаційної роботи, схеми архітектури, бази даних та алгоритмів роботи системи

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Спеціальна частина	*Коноваленко І.В.*		
Безпека життєдіяльності,	*доц. Сенчишин В.С.*		
основи хорони праці			

7. Дата видачі завдання *«____» _____20_____ р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи	Примітка
1.	*Аналітична частина*	*20.05.2026*	
2.	*Проєктна частина*	*25.05. 2026*	
3.	*Спеціальна частина*	*05.05. 2026*	
4.	*Безпека життєдіяльності, основи охорони праці*	*10.06. 2026*	
5.	*Оформлення графічної частини та пояснювальної*	*20.06. 2026*	
	записки		
6.	*Захист кваліфікаційної роботи*	*26.06. 2026*	

Студент _____ *Кіндій Д.Р.*
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи _____ *Коноваленко І.В.*
(підпис) (прізвище та ініціали)

ЗМІСТ

АНОТАЦІЯ

У кваліфікаційній роботі розроблено персонального AI-асистента для підтримки процесу пошуку вакансій, ведення профілю кандидата, обробки CV, автоматизованого збору вакансій з різних джерел та їх подальшої інтелектуальної оцінки. Основна увага приділена побудові системи, яка не обмежується пошуком за ключовими словами, а використовує великі мовні моделі для змістового аналізу назви вакансії, короткого опису, повного тексту вакансії та профілю конкретного кандидата.

Програмний продукт реалізовано як вебзастосунок на основі Django. Для збереження даних використано PostgreSQL, для семантичного пошуку - pgvector та ембединг-подання вакансій, для AI-аналізу - OpenAI API, а для організації чат-взаємодії - LangChain. Запуск і розгортання застосунку виконано за Docker-first підходом, що забезпечує відтворюваність середовища та спрощує перенесення системи на сервер.

Робота складається з п'яти основних розділів. У першому розділі проаналізовано предметну область, існуючі підходи та сформовано технічне завдання. У другому розділі описано архітектуру, модель даних та алгоритми роботи системи. Третій розділ містить інструкції з розгортання й експлуатації. Четвертий розділ присвячено економічному обґрунтуванню розробки. У п'ятому розділі розглянуто питання охорони праці, безпеки роботи з персональним комп'ютером та захисту даних.

Практична цінність роботи полягає у створенні працездатного програмного застосунку, який може використовуватися як персональний інструмент кандидата для систематизації пошуку роботи та як основа для подальшого розвитку інтеграцій, зокрема нотифікацій через Gmail.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

AI — штучний інтелект, технології та програмні методи, що дають змогу автоматизовано аналізувати текстові дані, формувати висновки та підтримувати прийняття рішень.

LLM — велика мовна модель, що використовується для аналізу природної мови, оцінювання відповідності вакансій профілю кандидата та формування пояснень у чат-інтерфейсі.

CV — резюме кандидата, яке завантажується в систему, перетворюється на текст і використовується для подальшого семантичного пошуку та порівняння з вакансіями.

API — програмний інтерфейс взаємодії між компонентами системи або зовнішніми сервісами.

SMTP — протокол передавання електронної пошти, використаний у роботі для надсилання Gmail-сповіщень про релевантні вакансії.

OAuth — механізм авторизації доступу до зовнішніх сервісів, який може бути використаний у подальшому розширенні інтеграції з Google Workspace.

pgvector — розширення PostgreSQL для зберігання векторних представлень тексту та виконання семантичного пошуку.

Embedding — векторне представлення тексту, яке дає змогу порівнювати CV, профіль кандидата й вакансії за змістом, а не лише за ключовими словами.

Django — серверний вебфреймворк, використаний для реалізації бізнес-логіки, моделей даних, адміністративної частини та API системи.

Docker — платформа контейнеризації, що забезпечує відтворюваний запуск застосунку разом із залежностями.

LangChain — бібліотека для побудови ланцюжків роботи з мовними моделями, семантичним пошуком і запитами природною мовою.

Gmail SMTP — механізм надсилання поштових повідомлень через обліковий запис Gmail з використанням пароля застосунку.

ВСТУП

Пошук роботи у сучасних умовах дедалі більше залежить від швидкості опрацювання інформації. Кандидат стикається з великою кількістю вакансій, різними форматами опису вимог, дублюванням оголошень, неточними фільтрами та необхідністю постійно співставляти свій досвід із потребами роботодавців. Якщо цей процес виконувати вручну, він займає багато часу та часто призводить до пропуску релевантних пропозицій або до зайвого перегляду вакансій, які не відповідають профілю кандидата [1–5].

Типові джоб-борди пропонують пошук за ключовими словами, фільтри за містом, форматом роботи, рівнем досвіду та заробітною платою. Проте такий підхід має обмеження. Вакансія може бути релевантною навіть тоді, коли в ній не використано точні ключові слова з резюме кандидата. І навпаки, наявність потрібного слова в описі не гарантує реальної відповідності ролі. Через це виникає потреба у більш гнучкому семантичному аналізі.

Великі мовні моделі дають змогу аналізувати текст вакансії не як набір окремих слів, а як змістовний опис ролі, вимог і контексту роботи. Модель може враховувати професійний досвід, навички, бажаний напрям розвитку, обмеження кандидата та інші параметри. Це дозволяє створити персонального асистента, який діє не як звичайний пошуковий фільтр, а як помічник із пояснюваними рекомендаціями [6–9].

Метою роботи є розроблення персонального AI-асистента, який забезпечує збереження профілю кандидата, завантаження й аналіз CV, автоматичний збір вакансій з різних джерел, двоетапне AI-оцінювання релевантності, категоризацію результатів та пошук вакансій через чат природною мовою.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати предметну область персонального пошуку вакансій.

2. Дослідити можливості використання великих мовних моделей для оцінки вакансій.

3. Обґрунтувати вибір технологій для серверної частини, бази даних, AI-модулів та розгортання.

4. Спроєктувати архітектуру системи та структуру даних.

5. Реалізувати модулі профілю кандидата, обробки CV, джерел вакансій, двоетапного сканування та чат-пошуку.

6. Забезпечити контейнеризований запуск системи та підготувати деплой-скрипти.

7. Провести тестування основних сценаріїв роботи.

Об'єктом дослідження є процес інтелектуальної підтримки пошуку вакансій та автоматизації персональних інформаційних процесів кандидата.

Предметом дослідження є методи й засоби побудови персонального AI-асистента на основі великих мовних моделей, семантичного пошуку, веб-сканування джерел вакансій та контейнеризованої серверної архітектури.

Практичне значення одержаних результатів полягає у створенні працездатного застосунку Personal Job Helper, який можна використовувати для ведення профілю кандидата, аналізу CV, пошуку релевантних вакансій та формування пояснюваних рекомендацій [10–15].

1 АНАЛІТИЧНИЙ РОЗДІЛ

У загальному розділі розглянуто предметну область, проаналізовано існуючі рішення та сформовано технічне завдання на розроблення системи. Структура цього розділу взята за логікою захищеного дипломного проєкту з архіву: спочатку подано аналітичний огляд, далі визначено призначення розробки, вимоги, техніко-економічні показники, етапи виконання та порядок приймання результатів [16–19].

1.1 Аналіз предметної області

Персональний пошук вакансій можна розглядати як інформаційний процес, що має повторюваний характер. Кандидат збирає дані про себе, підтримує актуальність резюме, переглядає нові вакансії, порівнює їх із власними навичками, відбирає перспективні варіанти та приймає рішення щодо відгуку. У цьому процесі наявні дані, правила, джерела інформації, критерії оцінки та результати, які потрібно зберігати для подальшого аналізу.

Основна складність полягає в тому, що вакансії описуються нерівномірно. Один роботодавець детально вказує технологічний стек, інший пише короткий маркетинговий опис, третій змішує вимоги, обов'язки й переваги в одному тексті. Через це кандидат змушений витрачати час на ручне прочитання навіть тих вакансій, які зрештою виявляються нерелевантними [20–23].

Додатковою проблемою є неоднозначність назв посад. Наприклад, позиції з подібним змістом можуть називатися по-різному: розробник, інженер, спеціаліст з автоматизації, AI-інженер, backend developer або data-oriented developer. Простий фільтр за назвою не завжди здатний знайти всі релевантні варіанти. Тому система має оцінювати не тільки буквальний збіг, а і зміст ролі.

Ще одним фактором є особистий контекст кандидата. Для одного користувача вакансія з певним стеком буде цікавою, для іншого - ні. Система повинна враховувати цільову позицію, рівень досвіду, навички, досягнення,

бажаний формат роботи, локаційні обмеження, очікування щодо компенсації та небажані домени. Саме тому персоналізація є центральною вимогою до розробки [24–26].

У межах даної роботи пошук вакансій розглядається не як загальний сервіс для всіх користувачів, а як приватний інструмент одного кандидата. Такий підхід дозволяє зберігати CV, історію пошуку та AI-оцінки в одному місці, а також використовувати ці дані для подальших рекомендацій.

1.2 Аналітичний огляд існуючих рішень

Існуючі рішення для пошуку роботи умовно можна поділити на кілька груп: джоб-борди, агрегатори вакансій, системи автоматичного відгуку, ATS-платформи для роботодавців, чат-боти та AI-помічники. Кожна група розв'язує частину задачі, але не закриває весь персональний сценарій кандидата.

Джоб-борди надають доступ до великої кількості вакансій і підтримують базові фільтри. Їх перевагою є широкий вибір джерел, проте недоліком є потреба в ручному перегляді та відсутність глибокої персональної оцінки. Користувач самостійно визначає, чи відповідає вакансія його досвіду.

Агрегатори вакансій збирають оголошення з кількох сайтів і спрощують пошук по різних джерелах. Проте вони часто дублюють вакансії, не завжди коректно витягують структуру опису та все одно працюють переважно з ключовими словами. Це зменшує якість рекомендацій для конкретного кандидата.

Системи автоматичного відгуку можуть прискорювати подачу заявок, але вони не розв'язують проблему якісного відбору. Якщо автоматизація працює без семантичного аналізу, кандидат може надсилати відгуки на нерелевантні вакансії, що знижує ефективність пошуку [27–29].

ATS-платформи призначені переважно для роботодавців і рекрутерів. Вони допомагають обробляти кандидатів, але не завжди зручні для самого кандидата, який хоче керувати власною стратегією пошуку, порівнювати вакансії та бачити пояснення відповідності.

AI-помічники й чат-боти можуть давати рекомендації, однак часто не мають власної бази вакансій, не зберігають історію сканувань і не інтегровані з приватним профілем кандидата. Тому в роботі запропоновано систему, яка поєднує локальне збереження даних, регулярний збір вакансій, LLM-аналіз і семантичний чат.

Порівняльний аналіз показує, що найкращий результат можна отримати не через один окремий інструмент, а через комбінацію кількох підходів: вебзастосунок для керування даними, фоновий збір вакансій, AI-скринінг, векторний пошук і чат для природної взаємодії [30–35].

1.3 Обґрунтування вибору технологій

	django	Flask	FastAPI
Type	Full-Stack Framework	Microframework	Asynchronous API Framework
Ease of Use	🟡 Moderate (structured)	🟢 Easy (flexible)	🟡 Moderate (requires async knowledge)
Performance	🔴 Moderate	🟡 High	🟢 Very High
Built-in Tools	🟢 Extensive	🔴 Minimal	🟡 Moderate
Best For	Large, complex web applications	Simple web apps and APIs	High-performance APIs and async apps
Learning Curve	🟡 Medium	🔴 Low	🟢 Medium to High
Community Support	🟢 Very Strong	🟡 Strong	🔴 Growing Rapidly

Django - має вбудовану ORM, систему автентифікації, адмін-панель, механізм форм і шаблонів; добре підходить для швидкого створення монолітного вебзастосунку з базою даних.

FastAPI - зручний для побудови API та асинхронних сервісів, але потребує додаткового вибору компонентів для шаблонів, авторизації, адмін-інтерфейсу та форм.

Flask - легкий і гнучкий, проте для задачі з профілями, формами, моделями, адмінкою та міграціями вимагає більше ручної збірки інфраструктури.

Laravel - є сильним вебфреймворком, але проєкт уже орієнтований на Python-екосистему, OpenAI SDK, LangChain і pypdf, тому Django краще узгоджується з обраним стеком.

Для реалізації обрано Django, оскільки він забезпечує достатньо повний набір інструментів для серверного застосунку: моделі, міграції, автентифікацію, форми, шаблони, маршрутизацію та адміністративний інтерфейс. Це дає змогу

зосередитися на бізнес-логіці AI-асистента, а не на ручному збиранні базових компонентів.

PostgreSQL обрано як основну базу даних через надійність, підтримку транзакцій, індексів, JSON-полів і розширень. Розширення pgvector дозволяє зберігати ембединг-представлення вакансій і виконувати семантичний пошук без введення окремої векторної бази даних. Це спрощує архітектуру та зменшує кількість сервісів у розгортанні.

OpenAI API використано для змістового аналізу вакансій, формування пояснень та генерації конфігурацій парсингу. LangChain використано в чат-модулі для класифікації наміру користувача та організації ланцюжка роботи із запитом. Docker Compose обрано для запуску всіх компонентів у єдиному відтворюваному середовищі.

1.4 Технічне завдання

Найменування розробки: персональний AI-асистент для пошуку вакансій Personal Job Helper. Область застосування: автоматизація персонального пошуку роботи, аналіз CV, відбір вакансій, збереження історії пошуку та формування рекомендацій [36–39].

Призначення розробки полягає у створенні вебзастосунку, який дозволяє користувачу вести власний профіль кандидата, завантажувати CV, підключати джерела вакансій, запускати двоетапне сканування, переглядати результати за категоріями та ставити запити до бази вакансій природною мовою.

До функціональних вимог належать: реєстрація або вхід користувача; редагування профілю кандидата; завантаження PDF-файлу CV; витягування тексту з CV; створення джерел вакансій; генерація конфігурацій парсингу; збір нових вакансій; AI-відбір за назвою і прев'ю; завантаження детальної сторінки;

розрахунок відсотка відповідності; категоризація; чат-пошук; перегляд історії сканувань.

До нефункціональних вимог належать: запуск у Docker; збереження даних у PostgreSQL; ізоляція профілю користувача; зберігання API-ключів у змінних середовища; зрозумілий темний інтерфейс; можливість подальшого розширення нотифікаціями; достатня простота деплою на VPS [40–41].

Техніко-економічні показники системи визначаються скороченням часу ручного перегляду вакансій, зменшенням кількості нерелевантних відкритих сторінок і підвищенням якості відбору через персоналізований AI-аналіз. Застосунок має практичну цінність як інструмент для одного кандидата та як основа майбутнього SaaS-рішення.

Стадії розробки включають аналіз вимог, проєктування архітектури, створення моделей даних, реалізацію профілю і CV, реалізацію парсингу, підключення AI-модулів, створення чат-інтерфейсу, тестування, підготовку Docker-розгортання та оформлення документації.

Порядок контролю та приймання передбачає перевірку запуску системи в Docker, створення тестового користувача, заповнення профілю, завантаження CV, додавання джерела вакансій, виконання сканування, перевірку AI-оцінки, категоризації та роботи чат-запитів.

2 ПРОЄКТНИЙ РОЗДІЛ

У цьому розділі подано опис інструментальних засобів реалізації, архітектури, бази даних, алгоритмів і користувацького інтерфейсу. Розділ відповідає основній проєктній частині дипломної роботи та пояснює, як саме побудовано програмний продукт.

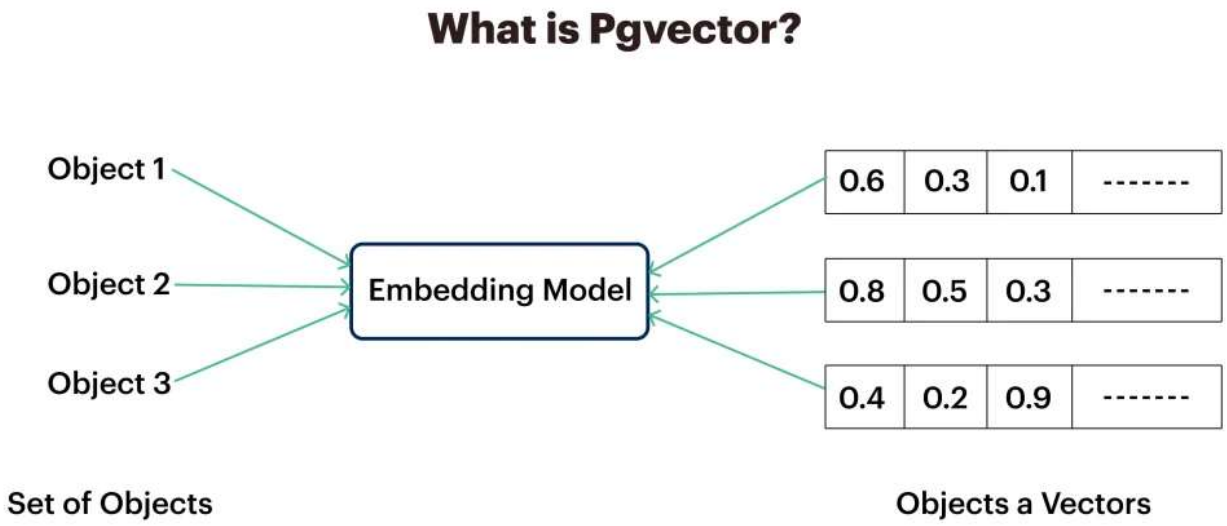
2.1 Інструментальні засоби реалізації

Python використано як основну мову програмування серверної частини. Його перевагами для даної задачі є широка екосистема бібліотек для веброзробки, обробки тексту, роботи з PDF, інтеграції з AI-сервісами та тестування.

Django забезпечує структуру застосунку. У проєкті він використовується для опису моделей, міграцій, форм, представлень, URL-маршрутів і шаблонів. Вбудована система автентифікації дозволяє обмежити доступ до профілю кандидата та його вакансій.

PostgreSQL використовується для збереження профілю кандидата, джерел вакансій, самих вакансій, журналів сканування, повідомлень чату та службових даних. Підтримка JSON-полів є зручною для збереження результатів AI-аналізу та сирих даних парсингу.

pgvector використовується для збереження векторних представлень вакансій. Завдяки цьому можна виконувати пошук за семантичною близькістю між текстом запиту та описами вакансій. У системі це застосовується для пошуку найкращих вакансій під CV і уточнювальні запити користувача.



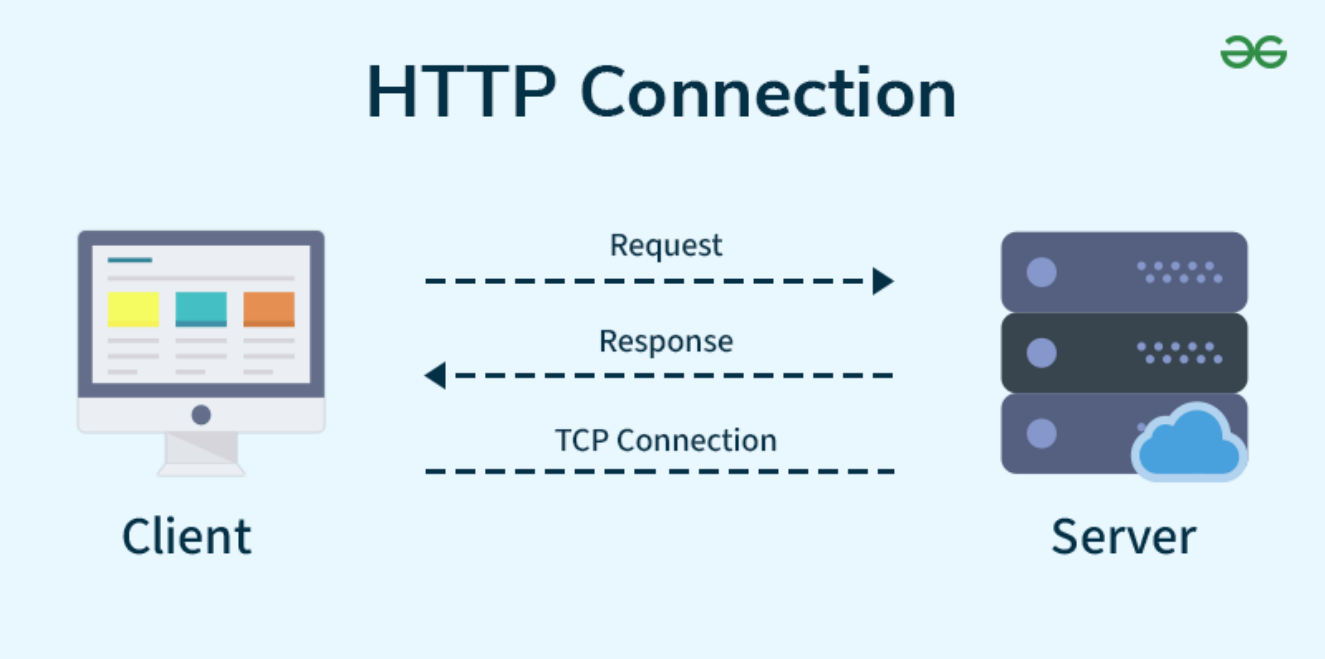
OpenAI API використовується для кількох задач: генерації конфігурацій парсингу, попереднього відбору вакансій, детального матчингу вакансії з профілем кандидата та формування відповідей у чаті. Такий розподіл дозволяє використовувати модель там, де потрібне змістове розуміння тексту.

LangChain використано для організації логіки чат-запитів. У системі він допомагає класифікувати намір користувача, відрізняти пошукові запити від статистичних, формувати промпт із профілем кандидата й передавати знайдені вакансії для пояснення.

Docker Compose описує сервіси застосунку: вебконтейнер, фоновий scheduler і PostgreSQL. Такий підхід дозволяє запускати систему однією командою та зменшує залежність від локального середовища розробника.

2.2 Архітектура програмної системи

Архітектура системи є модульною. Центральним компонентом є Django-застосунок, який обробляє HTTP-запити, працює з базою даних і координує взаємодію з AI-модулями. Окремий scheduler виконує періодичні задачі сканування джерел вакансій. База даних зберігає як структуровані дані, так і векторний індекс.



Користувач взаємодіє із системою через вебінтерфейс. Після входу він може заповнити профіль, завантажити CV, створити джерело вакансій, запустити сканування, переглянути знайдені вакансії та поставити питання у чаті. Усі дані прив'язуються до конкретного користувача.

Модуль профілю кандидата є джерелом персонального контексту для AI. Саме з нього модель отримує цільову позицію, досвід, навички, досягнення, очікування та обмеження. Якщо користувач завантажує CV, його текст також стає частиною контексту.

Модуль джерел вакансій зберігає URL, тип джерела, інтервал сканування, максимальну кількість вакансій за один запуск та конфігурацію парсингу. Конфігурація може бути задана вручну або згенерована AI на основі URL і типу джерела.

Модуль сканування реалізує двоетапну логіку. Перший етап збирає нові вакансії та передає AI тільки короткі прев'ю. Другий етап відкриває повну сторінку лише для вакансій, які модель визнала потенційно цікавими. Це зменшує кількість зайвих запитів і робить обробку більш керованою.

Модуль чату працює поверх накопиченої бази вакансій. Він не шукає інформацію в інтернеті напряму, а використовує вже зібрані та проаналізовані вакансії. Це дає кращий контроль над джерелами даних і дозволяє пояснювати рекомендації на основі фактичної бази системи.

2.3 Проєктування бази даних

Модель `CandidateProfile` реалізує один профіль на одного користувача. У ній зберігаються цільова позиція, рівень, побажання щодо локації та формату роботи, очікування щодо компенсації, навички, досвід, досягнення, обмеження, файл CV і витягнутий текст CV.

```
class CandidateProfile(models.Model):
    user = models.OneToOneField(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, related_name="candidate_profile")
    target_title = models.CharField(max_length=180, blank=True)
    seniority = models.CharField(max_length=80, blank=True)
    location_preferences = models.CharField(max_length=255, blank=True)
    work_preferences = models.CharField(max_length=255, blank=True)
    compensation_expectation = models.CharField(max_length=120, blank=True)
    skills_text = models.TextField(blank=True)
    experience_text = models.TextField(blank=True)
    achievements_text = models.TextField(blank=True)
    dealbreakers_text = models.TextField(blank=True)
    cv_file = models.FileField(upload_to="candidate_cvs/", blank=True)
    cv_original_filename = models.CharField(max_length=255, blank=True)
    cv_text = models.TextField(blank=True)
    ai_profile_summary = models.TextField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

Модель `JobSource` описує джерело вакансій. Вона містить назву, URL, тип джерела, ознаку активності, інтервал сканування, обмеження кількості вакансій, YAML-конфігурацію, згенерований JSON-конфіг і службові дати останнього та наступного сканування.

```python
class JobSource(models.Model):
    class SourceType(models.TextChoices):
        HTML = "html", "HTML page"
        RSS = "rss", "RSS/XML feed"
        JSON = "json", "JSON endpoint"

    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, related_name="job_sources")
    name = models.CharField(max_length=160)
    url = models.URLField()
    source_type = models.CharField(max_length=16, choices=SourceType.choices, default=SourceType.HTML)
    active = models.BooleanField(default=True)
    scan_interval_minutes = models.PositiveIntegerField(default=360)
    max_jobs_per_scan = models.PositiveIntegerField(default=50)
    config_yaml = models.TextField(blank=True)
    generated_config_json = models.JSONField(default=dict, blank=True)
    last_scan_at = models.DateTimeField(null=True, blank=True)
    next_scan_at = models.DateTimeField(default=timezone.now)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

Модель `Job` є центральною для збереження результатів. У ній містяться назва вакансії, компанія, локація, зовнішнє посилання, коротке прев'ю, повний опис, статус, AI-оцінки, відсоток відповідності, пояснення, сильні сторони, прогалини, категорія та дати обробки.

```python
class Job(models.Model):
    class Status(models.TextChoices):
        NEW = "new", "New"
        TITLE_REJECTED = "title_rejected", "Rejected by title"
        DETAIL_PENDING = "detail_pending", "Detail pending"
        MATCHED = "matched", "Matched"
        ARCHIVED = "archived", "Archived"
        ERROR = "error", "Error"

    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, related_name="jobs")
    source = models.ForeignKey(JobSource, on_delete=models.CASCADE, related_name="jobs")
    category = models.ForeignKey(JobCategory, on_delete=models.SET_NULL, null=True, blank=True, related_name="jobs")
    title = models.CharField(max_length=255)
    company = models.CharField(max_length=160, blank=True)
    location = models.CharField(max_length=160, blank=True)
    external_url = models.URLField(max_length=1200)
    preview_text = models.TextField(blank=True)
    description = models.TextField(blank=True)
    raw_json = models.JSONField(default=dict, blank=True)
    status = models.CharField(max_length=32, choices=Status.choices, default=Status.NEW)
    title_interest_score = models.PositiveSmallIntegerField(default=0)
    match_percent = models.PositiveSmallIntegerField(default=0)
    ai_reason = models.TextField(blank=True)
    ai_strengths = models.JSONField(default=list, blank=True)
    ai_gaps = models.JSONField(default=list, blank=True)
    first_seen_at = models.DateTimeField(auto_now_add=True)
    detail_fetched_at = models.DateTimeField(null=True, blank=True)
    matched_at = models.DateTimeField(null=True, blank=True)
```

Модель `JobCategory` дозволяє групувати вакансії в практичні категорії. Категорія створюється для конкретного користувача та може мати власний колір. Це робить перегляд вакансій зручнішим, особливо коли в базі накопичується багато результатів.

```python
class JobCategory(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, related_name="job_categories")
    name = models.CharField(max_length=80)
    description = models.TextField(blank=True)
    color = models.CharField(max_length=24, default="#7dd3fc")
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        unique_together = [("user", "name")]
        ordering = ["name"]

    def __str__(self) -> str:
        return self.name
```

Модель `ScanRun` зберігає дані про кожен запуск сканування. У ній фіксуються статус, етап, кількість нових вакансій, кількість відібраних AI вакансій, кількість оброблених детальних сторінок, повідомлення про помилки та журнал подій.

```python
class ScanRun(models.Model):
    class Status(models.TextChoices):
        RUNNING = "running", "Running"
        SUCCESS = "success", "Success"
        FAILED = "failed", "Failed"

    class ReviewStatus(models.TextChoices):
        NOT_REQUIRED = "not_required", "Not required"
        PENDING = "pending", "Pending review"
        ACCEPTED = "accepted", "Accepted"
        REJECTED = "rejected", "Rejected"
        REVISION_REQUESTED = "revision_requested", "Revision requested"

    source = models.ForeignKey(JobSource, on_delete=models.CASCADE, related_name="scan_runs")
    status = models.CharField(max_length=16, choices=Status.choices, default=Status.RUNNING)
    stage = models.CharField(max_length=64, default="stage_1")
    new_jobs_count = models.PositiveIntegerField(default=0)
    title_selected_count = models.PositiveIntegerField(default=0)
    detail_processed_count = models.PositiveIntegerField(default=0)
    config_snapshot = models.JSONField(default=dict, blank=True)
    preview_jobs_json = models.JSONField(default=list, blank=True)
    preview_jobs_count = models.PositiveIntegerField(default=0)
    operator_feedback = models.TextField(blank=True)
```

Моделі `JobChatSession` і `JobChatMessage` відповідають за історію чат-взаємодії. Повідомлення асистента можуть посилатися на конкретні вакансії, що дає змогу зберігати зв'язок між відповіддю моделі та фактичними даними з бази.

Окрема таблиця `assistant_app_job_embedding_index` використовується для векторного індексу. Вона містить ідентифікатор вакансії, ідентифікатор користувача, хеш контенту, вектор ембедингу та дату індексації. Хеш потрібен для того, щоб не переобчислювати ембединг, якщо текст вакансії не змінився.

2.4 Алгоритми роботи основних модулів

Алгоритм обробки CV починається із завантаження PDF-файлу. Система перевіряє тип файлу, витягує текст і зберігає його в полі профілю кандидата. Надалі цей текст використовується під час AI-скринінгу та формування семантичного запиту для пошуку вакансій.

Алгоритм генерації конфігурації джерела формує базову двоетапну структуру. Для першого етапу визначаються селектори списку вакансій, назви, посилання, компанії, локації й короткого прев'ю. Для другого етапу визначаються селектори заголовка, опису, компанії та локації на детальній сторінці.

Алгоритм першого етапу сканування завантажує сторінку або feed, витягує список вакансій, нормалізує посилання та зберігає тільки ті вакансії, яких ще немає в базі. Після цього нові вакансії передаються до AI-модуля у вигляді короткого списку з назвами та прев'ю.

AI-модуль першого етапу не застосовує жорсткі ключові слова. Він отримує профіль кандидата і список нових вакансій, після чого для кожної вакансії визначає, чи варто переходити до детальної сторінки. Результатом є ознака цікавості, оцінка інтересу та коротке пояснення.

Алгоритм другого етапу працює тільки з відібраними вакансіями. Для кожної з них завантажується повна сторінка, витягується детальний опис, після чого AI оцінює відповідність вакансії профілю кандидата. Результатом є відсоток відповідності, категорія, пояснення, сильні сторони та прогалини.

Алгоритм чат-пошуку починається з класифікації наміру. Якщо користувач просить кількість вакансій, система виконує фільтрацію через ORM. Якщо користувач просить знайти, порівняти або пояснити вакансії, система формує семантичний запит, оновлює ембедінги за потреби та виконує пошук через pgvector.

Після векторного пошуку найрелевантніші вакансії передаються до LLM разом із профілем кандидата й початковим запитом. Модель формує відповідь, пояснює логіку вибору і може запропонувати наступні запити. Це робить взаємодію більш природною, ніж перегляд статичного списку.

2.5 Інтерфейс користувача

Інтерфейс системи побудовано як темну адміністративно-прикладну панель із лівим боковим меню. Такий підхід підходить для інструмента, яким користувач може користуватися регулярно: меню завжди доступне, а основні розділи не приховані за зайвими переходами.

Головна панель показує стан профілю, кількість вакансій, останні результати та недавні сканування. Це дозволяє швидко зрозуміти, чи готовий профіль до AI-аналізу, чи є нові вакансії та коли востаннє запускалося сканування.

Сторінка профілю містить поля для цільової позиції, рівня, локаційних уподобань, формату роботи, очікуваної компенсації, навичок, досвіду, досягнень і обмежень. Також на цій сторінці реалізовано завантаження CV і перегляд витягнутого тексту.

Сторінка джерел вакансій дозволяє додавати нові джерела, задавати URL, тип, інтервал сканування та максимальну кількість вакансій за один запуск. Користувач може дозволити системі згенерувати конфігурацію або ввести її вручну.

Сторінка вакансій дає змогу переглядати знайдені пропозиції, їх статус, категорію, компанію, локацію та match percent. Детальна сторінка вакансії містить AI-пояснення, сильні сторони, прогалини та повний опис.

Chat-інтерфейс реалізовано як окремий розділ. Користувач може ставити запити природною мовою, отримувати список рекомендованих вакансій і бачити пояснення, чому саме ці вакансії вважаються найкращими.

2.6 Тестування програмної системи

Тестування системи охоплює модулі обробки CV, чат-логіку та фільтрацію вакансій. Для цього використано `pytest` і `pytest-django`. Наявність автоматизованих тестів важлива, оскільки частина логіки працює з датами, фільтрами, текстовими умовами та різними режимами запитів.

Тест обробки CV перевіряє, що система коректно реагує на PDF-файл і може витягнути текст для подальшого збереження. Це важливо, тому що CV є одним із головних джерел даних для AI-аналізу.

Тести чат-модуля перевіряють виділення пошукових токенів, визначення режиму запиту та застосування фільтрів. Наприклад, система повинна розпізнати запит про кількість Python-вакансій за останні два дні та обмежити результати відповідним часовим діапазоном.

Окремо перевіряється фільтрація за remote-режимом, навичками та винятками. Такі сценарії є важливими для природних запитів користувача, наприклад: знайди вакансії як моє CV, але без fintech.

Функціональне ручне тестування включає запуск Docker Compose, створення користувача, заповнення профілю, завантаження CV, створення джерела вакансій, запуск сканування, перевірку категоризації та роботу чат-запитів.

2.7 Деталізація програмних модулів

Модуль моделей даних є основою всієї системи. Він задає структуру профілю кандидата, джерел вакансій, категорій, самих вакансій, журналів

сканування та повідомлень чату. Завдяки використанню ORM розробник працює з сутностями на рівні Python-класів, а Django відповідає за створення таблиць, зв'язків і міграцій.

Модуль форм відповідає за приймання даних від користувача. Він використовується для редагування профілю кандидата, додавання джерела вакансій та введення повідомлення в чаті. Валідація форм важлива, оскільки система працює з URL, файлами CV, текстовими полями та числовими параметрами сканування.

Модуль представлень координує взаємодію між користувачем, формами, сервісами та шаблонами. Саме він визначає, що відбувається після збереження профілю, запуску сканування або надсилання повідомлення в чаті. Представлення ізолюють HTTP-логіку від бізнес-логіки, яка винесена в окремі сервіси.

Модуль сервісів реалізує прикладну логіку, яку не варто розміщувати безпосередньо в представленнях. До нього належать створення профілю за потреби, збереження джерела з конфігурацією, запуск сканування, оновлення статусів вакансій, застосування результатів AI-матчингу та створення категорій.

Модуль AI містить функції, що взаємодіють з OpenAI API. Він формує вхідні дані для моделі, задає системні інструкції, очікує відповідь у форматі JSON та перетворює її на структури, придатні для збереження в базі даних. Окрема увага приділяється обмеженню значень match percent і нормалізації списків сильних сторін та прогалин.

Модуль векторного пошуку будує текстове представлення вакансії, обчислює хеш контенту, перевіряє актуальність ембедингу та записує вектор у таблицю pgvector. Якщо вакансія не змінилася, ембединг повторно не обчислюється, що зменшує кількість зайвих API-викликів.

Модуль чату поєднує кілька типів логіки: збереження історії, класифікацію наміру, застосування фільтрів, семантичний пошук, формування відповіді та прив'язку знайдених вакансій до повідомлення асистента. Такий підхід дозволяє користувачу повертатися до старих чат-сесій і бачити, які вакансії були рекомендовані.

Модуль нотифікацій відокремлено від основного матчингу. Це важливо для розширення системи, оскільки повідомлення можна надсилати різними каналами без зміни логіки аналізу вакансій. У реалізації передбачено Telegram-канал і Gmail-канал через SMTP, що дозволяє отримувати листи про релевантні вакансії після завершення AI-матчингу.

Модуль шаблонів формує візуальну частину застосунку. Він реалізує базовий layout, бокове меню, dashboard, сторінки профілю, джерел вакансій, списку вакансій, деталей вакансії, історії сканувань і чату. Використання серверних шаблонів спрощує розробку та зменшує кількість клієнтської логіки.

Модуль тестів підтверджує коректність окремих частин системи. Автоматизовані тести особливо важливі для логіки, яка може непомітно змінювати поведінку: парсинг CV, визначення наміру запиту, фільтрація вакансій та обробка дат.

2.8 Деталізація AI-логіки та промптів

AI-логіка системи поділена на кілька сценаріїв, кожен із яких має власну задачу. Такий поділ важливий, оскільки один універсальний промпт для всіх задач був би менш керованим. Натомість система окремо формує запит для конфігурації парсингу, окремо для первинного скринінгу, окремо для детального матчингу та окремо для чат-відповідей.

Під час генерації конфігурації модель отримує назву джерела, URL, тип джерела та короткий контекст профілю кандидата. Від моделі очікується проста

структура з двома етапами. У промпті прямо зазначено, що не потрібно створювати keyword-фільтри, оскільки відбір має виконувати AI, а не жорсткі правила.

Під час первинного скринінгу модель отримує список нових вакансій. Для кожної вакансії передається ідентифікатор, назва, компанія, локація та короткий опис. Це дає змогу швидко вирішити, чи варто відкривати повну сторінку. Такий підхід економить запити до джерела й зменшує обсяг тексту, який потрапляє в детальний аналіз.

Під час детального матчингу модель отримує повний профіль кандидата і повний опис вакансії. Вона повертає відсоток відповідності, категорію, пояснення, сильні сторони та прогалини. Ці дані зберігаються в базі й показуються користувачу на сторінці вакансії.

У чат-модулі перший AI-виклик класифікує намір. Це важливо, бо запити користувача можуть бути різними: одні потребують семантичного пошуку, інші - звичайного підрахунку. Наприклад, фраза «скільки вакансій по Python за останні 2 дні» не повинна запускати ранжування вакансій, а має виконати фільтрацію й повернути кількість.

Другий AI-виклик у чаті використовується для пояснення результатів пошуку. Після того як система знаходить релевантні вакансії через pgvector, вона передає їх моделі разом із профілем кандидата й початковим запитом. Модель не вигадує вакансії, а працює тільки з переданим набором даних.

Обмеження відповіді JSON-форматом у багатьох AI-викликах підвищує передбачуваність системи. Замість того щоб парсити довільний текст, застосунок очікує конкретні поля. Це зменшує ризик помилок при збереженні результатів і робить систему простішою для тестування.

2.9 Обробка помилок і граничних ситуацій

Під час роботи з зовнішніми джерелами вакансій можливі помилки мережі, зміна HTML-структури сторінки, недоступність сайту, дублювання посилань або відсутність потрібних полів. Система повинна фіксувати такі ситуації в журналі сканування, щоб користувач міг зрозуміти, на якому етапі виникла проблема.

Якщо OpenAI API key не заданий, система не повинна повністю ламатися. У такому випадку AI-функції можуть повертати fallback-результати або повідомлення про те, що AI-аналіз недоступний. Це важливо для локального запуску, демонстрації інтерфейсу й тестування базових сценаріїв без зовнішнього API.

Якщо користувач ще не заповнив профіль або не завантажив CV, система має показувати, що профіль не готовий до якісного AI-аналізу. Такий стан не є помилкою, але він впливає на якість рекомендацій, тому має бути явно відображений в інтерфейсі.

Якщо джерело вакансій повертає дублікати, система використовує унікальність за користувачем і зовнішнім URL. Це запобігає повторному збереженню однієї й тієї самої вакансії та зменшує шум у результатах.

Якщо ембединг для вакансії вже існує й текст вакансії не змінився, повторне обчислення не виконується. Це важливо для економії API-викликів і прискорення чат-пошуку. Для визначення змін використовується хеш текстового представлення вакансії.

Якщо чат-запит не дає результатів, система повертає користувачу зрозуміле повідомлення і пропонує розширити пошук або запустити нове сканування. Така поведінка краща, ніж порожній екран або технічна помилка.

2.10 Захист даних і приватність

Система працює з чутливими даними: CV, описом досвіду, очікуваннями щодо роботи, обмеженнями та історією перегляду вакансій. Тому приватність є однією з важливих нефункціональних вимог.

Доступ до даних обмежується автентифікацією. Профіль кандидата, джерела вакансій, категорії, вакансії та чат-сесії прив'язані до конкретного користувача. Це запобігає випадковому змішуванню даних між різними обліковими записами.

API-ключі не повинні зберігатися в коді. Для цього використовуються змінні середовища та `.env`-файл, який не повинен потрапляти до публічного репозиторію. Такий підхід відповідає загальним практикам безпечного розгортання.

Файли CV зберігаються в media-сховищі застосунку. Для продуктивного розгортання потрібно обмежувати доступ до цього каталогу через вебсервер і не віддавати приватні файли без перевірки прав доступу.

Взаємодія з зовнішніми AI-сервісами означає, що частина тексту профілю та вакансій передається до API. Тому в майбутньому доцільно додати налаштування рівня приватності, логування факту AI-запитів і можливість очищення профілю або CV за запитом користувача.

2.11 Сценарії роботи системи на рівні даних

Перший наскрізний сценарій починається з реєстрації або входу користувача. Після цього система перевіряє, чи існує профіль кандидата, і за потреби створює його. Така поведінка спрощує перший запуск, оскільки користувач не має окремо ініціалізувати профіль перед заповненням форми.

Після редагування профілю дані зберігаються в таблиці профілю кандидата. Якщо користувач завантажив PDF-файл, система витягує з нього текст і записує

його в окреме поле. Це дозволяє надалі працювати з текстом CV без повторного читання файлу.

Коли користувач створює джерело вакансій, система зберігає його базові параметри: назву, URL, тип, інтервал сканування та максимальну кількість вакансій. Якщо користувач обирає автоматичне формування конфігурації, AI-модуль створює структуру для першого й другого етапів парсингу.

Під час запуску сканування створюється запис `ScanRun`. Він є журналом конкретної спроби обробки джерела. У процесі роботи до нього додаються події: початок сканування, кількість зібраних нових вакансій, кількість відібраних AI вакансій, кількість оброблених детальних сторінок або текст помилки.

Якщо вакансія вже існує для користувача за тим самим зовнішнім URL, вона не створюється повторно. Це дозволяє запускати сканування регулярно без засмічення бази дублікатами. Нова вакансія отримує статус `new`, а після первинного AI-аналізу переходить у статус `detail_pending` або `title_rejected`.

Після другого етапу вакансія отримує статус `matched`. У цей момент вона має повний опис, відсоток відповідності, категорію, пояснення, сильні сторони та прогалини. Ці дані використовуються і в інтерфейсі, і в чаті, і в майбутніх нотифікаціях.

Коли користувач відкриває чат і ставить запит, система створює повідомлення користувача, класифікує намір, вибирає вакансії, формує відповідь асистента та зберігає її в історії. Якщо відповідь посилається на конкретні вакансії, між повідомленням і вакансіями створюється зв'язок.

2.12 Проєктування користувацьких екранів

Екран входу має бути мінімальним і зрозумілим. Його задача - надати доступ до особистого кабінету, не перевантажуючи користувача зайвою інформацією. Після входу користувач потрапляє на головну панель.

Головна панель виконує роль короткого стану системи. На ній доцільно показувати готовність профілю до AI-аналізу, кількість вакансій у базі, кількість релевантних вакансій, останні сканування та швидкі переходи до основних дій.

Екран профілю є одним із найважливіших, оскільки від якості профілю залежить якість рекомендацій. Поля мають бути сформульовані так, щоб користувач міг описати не лише навички, а й реальний досвід, досягнення, очікування та обмеження.

Екран джерел вакансій повинен допомагати користувачу зрозуміти, які сайти або feeds відстежуються. Для кожного джерела корисно показувати статус активності, інтервал сканування, дату останнього запуску та дату наступного запуску.

Екран деталей сканування має показувати не лише фінальний результат, а й журнал подій. Це важливо для діагностики: користувач бачить, скільки вакансій було знайдено, скільки з них AI відібрав і чи були помилки під час другого етапу.

Екран списку вакансій повинен бути придатним для швидкого перегляду. Для цього виводяться назва, компанія, локація, категорія, статус і match percent. Вакансії з високим відсотком відповідності мають бути помітними, бо саме вони є головним результатом системи.

Екран детальної вакансії має пояснювати рішення AI. Користувачу недостатньо бачити число, він повинен розуміти, чому вакансія отримала саме такий відсоток відповідності. Тому важливими є `ai_reason`, `ai_strengths` і `ai_gaps`.

Екран чату повинен підтримувати історію сесій. Це дозволяє користувачу повертатися до попередніх пошукових сценаріїв: наприклад, окремо мати чат для Python-вакансій, окремо для remote-позицій і окремо для порівняння кількох найкращих варіантів.

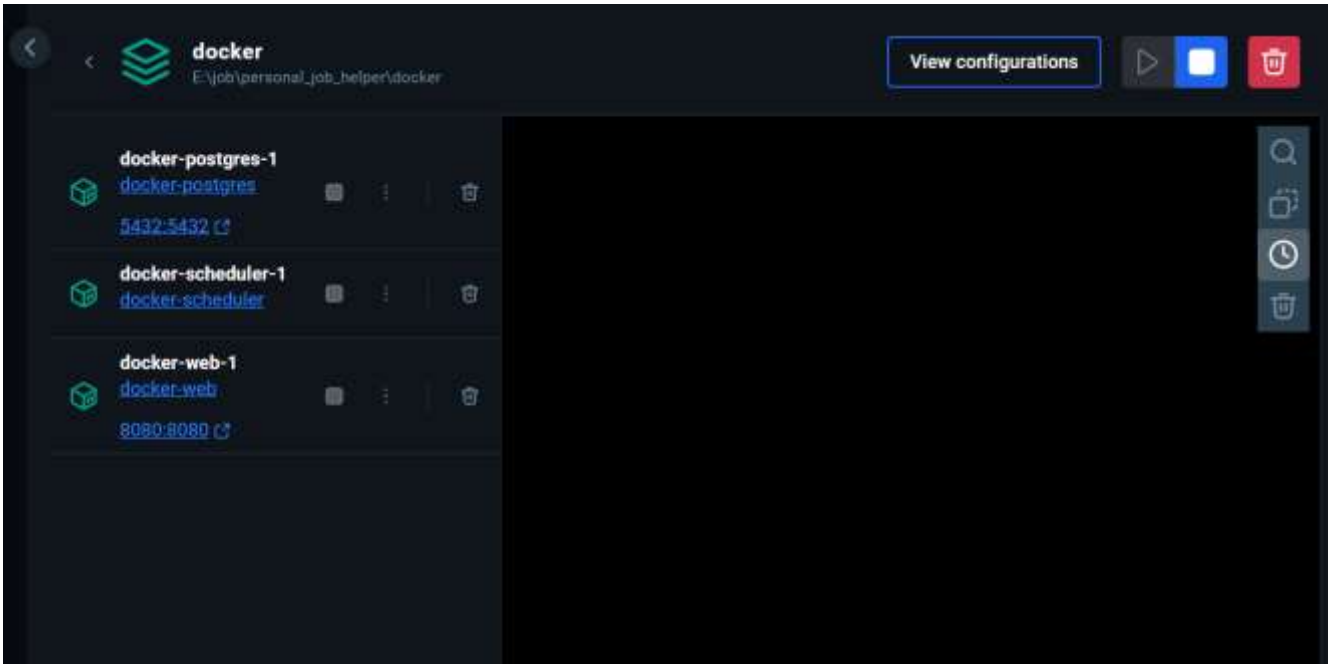
2.13 Контейнеризація та середовище виконання

Контейнеризація вирішує проблему відмінностей між локальним середовищем розробника та сервером. Усі ключові залежності описані в Dockerfile і docker-compose конфігурації, тому застосунок можна підняти однаково на різних машинах.

Сервіс `web` відповідає за запуск Django-застосунку. Перед стартом він виконує збір статичних файлів і міграції бази даних. Після цього запускається Gunicorn, який приймає HTTP-запити від вебсервера або напряму в локальному середовищі.

Сервіс `scheduler` використовує той самий код, що й web, але запускає іншу команду - фоновий цикл перевірки джерел вакансій. Це дозволяє не змішувати HTTP-обробку і періодичні задачі в одному процесі.

Сервіс `postgres` створено на базі окремого Dockerfile, у якому встановлюється pgvector. Це важливо, тому що стандартний PostgreSQL без розширення не зможе зберігати векторні представлення вакансій і виконувати пошук за відстанню.



Тома Docker використовуються для збереження даних бази та media-файлів. Це дозволяє перезапускати контейнери без втрати даних. Для продуктивного середовища потрібно також налаштувати резервне копіювання цих томів.

Змінні середовища виносяться в окремий файл. Такий підхід дає змогу мати різні налаштування для локального запуску, тестового сервера й продуктивного сервера без зміни коду.

2.14 Деплой і супровід системи

Для розгортання системи на сервері підготовлено скрипти, подібні за ідеєю до скриптів у проєктах сканерів. Вони автоматизують повторювані дії та зменшують ризик ручної помилки під час оновлення.

Скрипт початкової підготовки сервера створює користувача, робочу директорію та базові права доступу. Це дозволяє не запускати застосунок від імені root і розділити системні та прикладні файли.

Скрипт налаштування віртуального хоста готує конфігурацію вебсервера. У реальному розгортанні саме вебсервер має приймати зовнішній трафік, обробляти HTTPS і проксувати запити до контейнера застосунку.

Скрипт деплою виконує оновлення коду, збирання контейнерів, міграції та перезапуск сервісів. Це дає змогу швидко оновлювати застосунок після внесення змін у код.

Для супроводу системи важливо контролювати логи web і scheduler. Логи web допомагають знаходити помилки HTTP-запитів, а логи scheduler - проблеми зі скануванням джерел, AI-викликами або мережевими помилками.

Окремої уваги потребує контроль вартості AI-викликів. Оскільки система може обробляти багато вакансій, важливо обмежувати кількість вакансій за один запуск і використовувати перший етап відбору, щоб не передавати повний текст кожної вакансії до моделі.

2.15 Порівняння реалізованого підходу з пошуком за ключовими словами

Пошук за ключовими словами простий у реалізації, але він не враховує контекст. Наприклад, вакансія може містити слово Python у списку другорядних технологій, хоча фактично роль є Java-орієнтованою. Простий фільтр може помилково вважати її релевантною.

Інша ситуація виникає тоді, коли релевантна вакансія не містить точного слова з профілю кандидата. Наприклад, кандидат має досвід автоматизації бізнес-процесів, а вакансія описує інтеграції, робочі процеси, потоки даних або внутрішні інструменти. За змістом вакансія може підходити, але ключове слово може не збігтися.

AI-скринінг на першому етапі аналізує назву й прев'ю як змістовний опис. Він може врахувати суміжність ролей, рівень позиції, домен і потенційну відповідність профілю. Це не гарантує ідеальну точність, але зменшує залежність від ручних правил.

Детальний AI-матчинг на другому етапі працює вже з повним описом вакансії. Саме на цьому кроці система може пояснити, які вимоги збігаються з досвідом кандидата, які навички є сильними сторонами, а які можуть бути прогалинами.

Семантичний чат доповнює цей підхід. Користувач може формулювати запити мовою, близькою до природної: без fintech, remote only, схоже на моє CV, найкращі за останні дні. Це робить систему ближчою до персонального помічника, ніж до звичайного фільтра.

3 СПЕЦІАЛЬНА ЧАСТИНА

Спеціальний розділ містить інструкції з розгортання та експлуатації системи. Його призначення - показати, як практично запустити розроблений продукт і як користувач має працювати з основними можливостями.

3.1 Інструкція з розгортання системи

Для запуску системи необхідно мати встановлені Docker і Docker Compose. Усі основні сервіси описано у файлі `docker/docker-compose.yml`. Перед запуском потрібно створити файл змінних середовища на основі прикладу `.env.example`.

Основні змінні середовища включають параметри бази даних, порт застосунку, секретний ключ Django, налаштування OpenAI API та параметри нотифікацій. OpenAI API key є обов'язковим для повноцінного AI-аналізу, генерації конфігурацій та роботи чат-модуля.

Після налаштування середовища система запускається через Docker Compose. Контейнер PostgreSQL ініціалізує базу даних, контейнер web виконує міграції та запускає вебзастосунок, а контейнер scheduler відповідає за фонові запуски сканування.

Для першого входу необхідно створити адміністратора або тестового користувача. У попередньому налаштуванні для тестування використовувався користувач `admin` з паролем `admin`, однак для реального розгортання такий пароль потрібно змінити.

Деплой на сервер виконується за допомогою підготовлених скриптів. Скрипт підготовки користувача створює робоче середовище на VPS, скрипт налаштування віртуального хоста готує конфігурацію вебсервера, а скрипт деплою оновлює код, будує контейнери, виконує міграції та перезапускає сервіси.

3.2 Інструкція користувача

Після входу в систему користувач переходить до розділу профілю. На цьому етапі потрібно заповнити цільову позицію, рівень, бажаний формат роботи, локаційні побажання, навички, досвід, досягнення та обмеження. Чим повніший профіль, тим якісніше AI зможе оцінювати вакансії.

Наступним кроком є завантаження CV у форматі PDF. Система витягує текст із файлу й зберігає його у профілі. Користувач може переглянути витягнутий текст і за потреби оновити файл.

Після заповнення профілю користувач додає джерело вакансій. Для цього потрібно вказати назву джерела, URL, тип джерела та параметри сканування. Якщо користувач не хоче вручну описувати селектори, він може дозволити системі згенерувати конфігурацію.

Після запуску сканування система збирає нові вакансії, виконує перший AI-відбір, завантажує детальні сторінки для відібраних вакансій і розраховує відсоток відповідності. Результати можна переглядати у списку вакансій або за категоріями.

У чаті користувач може ставити питання природною мовою. Наприклад, можна попросити знайти найкращі вакансії під останнє CV, виключити певний домен або порахувати кількість вакансій за конкретною навичкою за останні дні.

3.3 Експлуатаційні сценарії

Перший сценарій - початкове налаштування профілю. Користувач входить у систему, заповнює основні дані, завантажує CV та перевіряє, що система зберегла текст резюме. Після цього профіль стає готовим до AI-аналізу.

Другий сценарій - підключення джерела вакансій. Користувач додає URL джоб-борда, задає тип джерела та інтервал сканування. Система генерує або приймає конфігурацію парсингу та планує наступний запуск.

Третій сценарій - аналіз нових вакансій. Scheduler або користувач запускає сканування. Система збирає нові вакансії, відсіює нецікаві на першому етапі, детально аналізує релевантні й записує результати в базу.

Четвертий сценарій - перегляд результатів. Користувач відкриває список вакансій, бачить match percent, категорію, пояснення та прогалини. За потреби він переходить на зовнішню сторінку вакансії.

П'ятий сценарій - чат-пошук. Користувач формулює запит у довільній формі. Система визначає, чи потрібен пошук або підрахунок, отримує дані з бази й формує відповідь з поясненням.

3.4 Розділ під Gmail-інтеграцію

У межах роботи реалізовано просту Gmail-інтеграцію для надсилання email-сповіщень про вакансії, які отримали достатньо високий відсоток відповідності профілю кандидата. Інтеграція виконана через Gmail SMTP та app password, без повного OAuth-сценарію. Такий варіант обрано як мінімально достатній для практичної задачі нотифікацій і демонстрації зв'язку системи з Google Workspace.

Gmail-сповіщення підключено до загального механізму `send_match_alert`. Після завершення другого етапу аналізу вакансії система зберігає результат AI-матчингу, визначає категорію, відсоток відповідності, сильні сторони та прогалини. Після цього викликається модуль нотифікацій, який незалежно перевіряє доступність Telegram і Gmail каналів.

Для Gmail-каналу використовуються такі параметри середовища: `GMAIL_SMTP_USERNAME`, `GMAIL_SMTP_APP_PASSWORD`,

`GMAIL_ALERT_RECIPIENT`, `GMAIL_ALERT_FROM_NAME` та `GMAIL_ALERT_MIN_MATCH_PERCENT`. Перші два параметри визначають обліковий запис Gmail, з якого надсилається лист. Одержувач може бути вказаний явно або братися з email користувача. Порогове значення дозволяє не надсилати листи для слабких збігів.

Лист містить назву вакансії, компанію, локацію, категорію, відсоток відповідності, посилання на вакансію, пояснення AI, сильні сторони та прогалини. Повідомлення формується у двох форматах: звичайний текст і HTML-версія. HTML-дані екрануються, оскільки назва вакансії, компанія або опис можуть походити із зовнішнього сайту.

Для перевірки працездатності інтеграції було створено app password у Google Account, додано Gmail-змінні до `.env` і виконано тестову відправку листа через `smtp.gmail.com` на порті 465 з SSL. Тестова відправка завершилася успішно, що підтвердило коректність облікових даних і можливість використання Gmail як каналу сповіщень.

Обраний SMTP-підхід має обмеження: він підходить для персонального використання або невеликого прототипу, але для багатокористувацького продукту доцільно перейти на OAuth-авторизацію Google та Gmail API. Проте для поточної задачі персонального AI-асистента SMTP-варіант є достатнім, простим у налаштуванні та зрозумілим для демонстрації на захисті.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Економічний розділ призначений для орієнтовного оцінювання витрат на розроблення програмного продукту та визначення доцільності виконання роботи. Розрахунки мають навчальний характер і можуть бути уточнені відповідно до методичних рекомендацій кафедри.

4.1 Визначення стадій технологічного процесу

Розроблення системи поділяється на кілька стадій: аналіз вимог, проєктування, розроблення серверної частини, реалізація AI-модулів, розроблення інтерфейсу, тестування, підготовка Docker-розгортання та оформлення документації.

Орієнтовна тривалість етапів така: аналіз предметної області - 20 годин, проєктування архітектури - 30 годин, розроблення моделей і бізнес-логіки - 80 годин, реалізація AI-модулів - 60 годин, інтерфейс - 40 годин, тестування - 30 годин, документація - 30 годин. Загальна тривалість становить близько 290 годин.

У роботі беруть участь умовні ролі: керівник, програміст і тестувальник. Для дипломного проєкту ці ролі можуть виконуватися студентом за консультаційної участі керівника, але в економічному розрахунку їх доцільно розділити для оцінки вартості робіт.

4.2 Розрахунок витрат на оплату праці

Для орієнтовного розрахунку приймаємо погодинні ставки: керівник - 180 грн/год, програміст - 220 грн/год, тестувальник - 160 грн/год. Розподіл годин: керівник - 30 годин, програміст - 220 годин, тестувальник - 40 годин.

Витрати на оплату праці керівника становлять 30 * 180 = 5400 грн. Витрати на оплату праці програміста становлять 220 * 220 = 48400 грн. Витрати на оплату

праці тестувальника становлять 40 * 160 = 6400 грн. Загальна сума основної заробітної плати становить 60200 грн.

Відрахування на соціальні заходи можна прийняти на рівні 22 відсотків. У такому випадку сума відрахувань становить 60200 * 0,22 = 13244 грн. Загальні витрати на оплату праці з відрахуваннями становлять 73444 грн.

4.3 Розрахунок витрат на електроенергію

Для розрахунку витрат на електроенергію приймаємо, що під час розроблення використовується персональний комп'ютер або ноутбук із середньою потужністю 0,12 кВт. Загальна тривалість активної роботи становить 290 годин.

Споживання електроенергії становить 0,12 * 290 = 34,8 кВт*год. За умовної вартості 4,32 грн за 1 кВт*год витрати на електроенергію становлять 34,8 * 4,32 = 150,34 грн.

Порівняно із витратами на оплату праці витрати на електроенергію є незначними, однак вони враховуються в загальній собівартості розробки.

4.4 Амортизаційні та накладні витрати

Для розрахунку амортизації приймаємо вартість робочого обладнання 30000 грн і строк корисного використання 36 місяців. Місячна амортизація становить 30000 / 36 = 833,33 грн.

Якщо розроблення тривало умовно два місяці, амортизаційні витрати становлять 1666,66 грн. До них можуть додаватися витрати на інтернет, хмарні сервіси, домен, сервер або API-виклики.

Накладні витрати приймаємо на рівні 15 відсотків від основної заробітної плати. У такому випадку вони становлять 60200 * 0,15 = 9030 грн.

4.5 Собівартість і економічна доцільність

Орієнтовна собівартість розробки складається з витрат на оплату праці з відрахуваннями, електроенергії, амортизації та накладних витрат. Сума становить 73444 + 150,34 + 1666,66 + 9030 = 84291 грн.

Якщо розглядати продукт як інструмент, що економить кандидату 5 годин щотижня протягом 6 місяців активного пошуку, економія часу становить приблизно 120 годин. Для фахівця з погодинною вартістю часу 300 грн це еквівалентно 36000 грн персональної економії тільки за один цикл пошуку.

Окрім прямої економії часу, система може підвищити якість відбору вакансій і зменшити кількість нерелевантних відгуків. Для комерційного розвитку продукт може бути основою сервісу з підпискою або внутрішнього інструмента рекрутингової автоматизації.

Таким чином, економічна доцільність розробки полягає не лише у безпосередній окупності, а й у створенні програмної основи, яку можна повторно використовувати, розширювати та адаптувати до інших сценаріїв AI-автоматизації.

5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ

Під час розроблення програмного забезпечення основна частина роботи виконується за персональним комп'ютером. Тому важливо враховувати вимоги до організації робочого місця, освітлення, режиму праці та відпочинку, електробезпеки й інформаційної безпеки.

5.1 Шкідливі та небезпечні чинники

До шкідливих чинників під час роботи програміста належать тривале зорове навантаження, статичне положення тіла, монотонність роботи, психоемоційне навантаження, недостатнє освітлення, шум і можливе порушення режиму праці та відпочинку.

Тривала робота перед монітором може спричиняти втому очей, головний біль і зниження концентрації. Неправильне розташування робочого місця може призвести до напруження м'язів шиї, спини та рук.

Психофізіологічні чинники пов'язані зі складністю задач, необхідністю тривалої концентрації, роботою з помилками та дедлайнами. Для зменшення їх впливу потрібно планувати роботу, робити перерви та чергувати типи завдань.

5.2 Організація робочого місця

Робоче місце має бути організоване так, щоб користувач міг працювати без зайвого напруження. Монітор слід розміщувати на відстані приблизно 50-70 см від очей, верхній край екрана має бути на рівні очей або трохи нижче.

Крісло повинно підтримувати спину, а ноги мають стояти на підлозі або підставці. Клавіатура й миша розміщуються так, щоб передпліччя були в природному положенні. Це зменшує ризик перенавантаження кистей і плечей.

Освітлення має бути достатнім і рівномірним. Бажано уникати відблисків на екрані та різкого контрасту між монітором і навколишнім середовищем. У приміщенні необхідно підтримувати нормальну вентиляцію та температуру.

5.3 Електробезпека та інформаційна безпека

Електрообладнання повинно бути справним, кабелі не мають створювати перешкод або ризику пошкодження. Для захисту обладнання доцільно використовувати мережеві фільтри або джерело безперебійного живлення.

Оскільки розроблена система працює з персональними даними кандидата та текстом CV, важливо забезпечити конфіденційність. Доступ до профілю має бути обмежений автентифікацією, а секретні ключі повинні зберігатися у змінних середовища, а не в коді.

Для безпечної експлуатації системи необхідно регулярно оновлювати залежності, використовувати складні паролі, обмежувати доступ до серверу, виконувати резервне копіювання бази даних і контролювати права доступу до файлів CV.

ВИСНОВКИ

У кваліфікаційній роботі розроблено персонального AI-асистента для пошуку вакансій, який поєднує збереження профілю кандидата, обробку CV, двоетапний AI-скринінг вакансій, категоризацію, семантичний пошук і чат-взаємодію природною мовою.

У процесі виконання роботи проаналізовано предметну область, розглянуто існуючі підходи, обґрунтовано вибір технологій, сформовано технічне завдання, спроєктовано архітектуру системи та реалізовано основні програмні модулі.

Розроблений застосунок використовує Django для серверної логіки, PostgreSQL для збереження даних, pgvector для семантичного пошуку, OpenAI API для AI-аналізу та LangChain для роботи чат-модуля. Docker Compose забезпечує відтворюваний запуск системи.

Практичним результатом роботи є система Personal Job Helper, яка дозволяє користувачу заповнити профіль, завантажити CV, підключити джерела вакансій, отримати AI-оцінку релевантності та ставити запити до бази вакансій природною мовою.

Подальший розвиток системи може включати перехід від Gmail SMTP до повного Gmail API з OAuth-авторизацією, розширення кількості джерел вакансій, покращення механізмів категоризації, додавання календарних нагадувань і формування персональних рекомендацій на основі історії відгуків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація Django. URL: https://docs.djangoproject.com/
2. Документація PostgreSQL. URL: https://www.postgresql.org/docs/
3. Документація pgvector. URL: https://github.com/pgvector/pgvector
4. Документація Docker. URL: https://docs.docker.com/
5. Документація OpenAI API. URL: https://platform.openai.com/docs/
6. Документація LangChain. URL: https://docs.langchain.com/
7. Документація Python. URL: https://docs.python.org/3/
8. Документація pypdf. URL: https://pypdf.readthedocs.io/
9. Документація httpx. URL: https://www.python-httpx.org/
10. Документація selectolax. URL: https://selectolax.readthedocs.io/
11. Матеріали кодової бази дипломного проєкту.
12. Коноваленко І. В. Платформа .NET та мова програмування C# 8.0 : навч. посіб. / І. В. Коноваленко, П. О. Марущак. – Тернопіль : В. А. Паляниця, 2020 – 320 с. http://library.megu.edu.ua:8180/jspui/handle/123456789/4115
13. Проектування мікропроцесорних систем керування: навчальний посібник / І.Р. Козбур, П.О. Марущак, В.Р. Медвідь, В.Б. Савків, В.П. Пісьціо. – Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2022. – 324 с. https://elartu.tntu.edu.ua/handle/lib/39189
14. Капаціла Ю.Б., Савків В.Б. Методичні вказівки до виконання кваліфікаційної роботи бакалавра спеціальності «Автоматизація, комп’ютерно-інтегровані технології та робототехніка». Тернопіль.: Видавництво ТНТУ. 2025. 60 с. https://elartu.tntu.edu.ua/handle/lib/48495
15. Капаціла Ю.Б., Шовкун О.П. Конструкторсько-технологічна практика. Методичні вказівки для здобувачів освітнього ступеня «бакалавр» спеціальності «Автоматизація, комп’ютерно-інтегровані технології та робототехніка». Тернопіль.: Видавництво ТНТУ. 2025. 22 с.
16. Козбур І.Р., Методичні вказівки до виконання лабораторної роботи «Дослідження частотних характеристик неперервних лінійних систем», по

курсу «Теорія автоматичного управління», для студентів 3 курсу спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / Авт.: Козбур І.Р., Козбур Г.В. Марущак П.О., Савків В.Б. – Тернопіль: ТНТУ, ФПТ, каф. АВ, – 2022. – с. 16. https://elartu.tntu.edu.ua/handle/lib/39207

17. Козбур І.Р., «Дослідження часових характеристик неперервних лінійних систем», по курсу «Теорія автоматичного управління», для студентів 3 курсу спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / Авт.: Козбур І.Р., Козбур Г.В. Марущак П.О., Савків В.Б. – Тернопіль: ТНТУ, ФПТ, каф. АВ, – 2022. – 19 с. https://elartu.tntu.edu.ua/handle/lib/39206
18. Автоматизація виробничих процесів. Навчальний посібник для технічних спеціальностей вищих навчальних закладів. / Я.І. Проць, В.Б. Савків, О.К. Шкодзінський, О.Л. Ляшук. Тернопіль: ТНТУ ім. І. Пулюя, 2011. 344 с. https://elartu.tntu.edu.ua/handle/123456789/1551
19. Методичні вказівки по роботі з програмним симулятором "AVR simulator IDE" з курсу "Мікропроцесорні та програмні засоби автоматизації" / укл. : В.Р. Медвідь , В.П. Пісьціо. - Тернопіль : ТНТУ імені Івана Пулюя, 2020. - 21 с.
20. Моделювання нелінійних систем керування у пакеті MATLAB SIMULINK, методичні вказівки до виконання лабораторної роботи по курсу «Комп’ютерні методи дослідження систем автоматичного управління», для студентів 4 курсу спеціальності 6.050201 «Системна інженерія» / укл. : І.Р. Козбур , Г.В. Козбур , Р.І. Михайлишин. – Тернопіль : ТНТУ імені Івана Пулюя, 2019. - 19 с. https://elartu.tntu.edu.ua/handle/lib/28057
21. Моделювання систем керування в пакеті MATLAB SIMULINK, методичні вказівки до виконання лабораторної роботи по курсу «Комп’ютерні методи дослідження систем автоматичного управління», для студентів 4 курсу спеціальності 6.050201 «Системна інженерія» / укл. : І.Р. Козбур , Г.В. Козбур , Р.І. Михайлишин. – Тернопіль : ТНТУ, 2019. - 23 с. https://elartu.tntu.edu.ua/handle/lib/28056
22. Проектування та аналіз електричних схем в програмному середовищі Multisim. Методичні вказівки до самостійної роботи студентів з курсу

"Проектування мікропроцесорних систем керування технологічними процесами" / укл. : В.Р. Медвідь , В.П. Пісьціо . - Тернопіль : ТНТУ Імені Івана Пулюя, 2018. - 26 с. https://elartu.tntu.edu.ua/handle/lib/26404

23. Проектування та аналіз електричних схем в програмному середовищі Proteus VSM. Методичні вказівки до самостійної роботи студентів з курсу "Проектування мікропроцесорних систем керування технологічними процесами" / укл. : В.Р. Медвідь , В.П. Пісьціо. - Тернопіль : ТНТУ, 2018. - 26 с. https://elartu.tntu.edu.ua/handle/lib/26397
24. Методичні вказівки до курсового проектування з курсу "Проектування систем автоматизації" / Савків В.Б., Шкодзінський О.К., Пісьціо В.П. Тернопіль : ТНТУ, 2025. 128 с. https://elartu.tntu.edu.ua/handle/lib/48646
25. Лабораторний практикум з проектування та моделювання роботи електропневматичних схем у середовищі програмного пакету «FluidSIM Pneumatics» з курсу «Технічні засоби автоматизації» / укл. : О.К. Шкодзінський. – Тернопіль : ТНТУ імені Івана Пулюя, 2020. - 32 с. https://elartu.tntu.edu.ua/handle/lib/31418
26. Трембач Р.Б., Медвідь В.Р. Методичні вказівки до виконання курсового проекту з дисципліни “Електроніка і мікросхемотехніка”. – Тернопіль: ТНТУ ім. І. Пулюя, 2024. – 53 с. https://elartu.tntu.edu.ua/handle/lib/44635
27. Трембач Р.Б., Медвідь В.Р. Методичні вказівки до виконання до виконання лабораторних робіт з дисципліни “Електроніка і мікросхемотехніка” Модуль 3. «Імпульсна техніка, вторинні джерела живлення, основи цифрової електроніки» – Тернопіль: ТНТУ, 2024. -26 с.
28. Трембач Р.Б., Шовкун О.П. Методичні вказівки до виконання до виконання лабораторних робіт з дисципліни “Інформаційно – вимірювальні системи” Модуль І. «Вимірювальна техніка» – Тернопіль: ТНТУ., 2024. – 67 с.
29. Аналіз стійкості і якості електромеханічної слідкуючої системи. : Методичні вказівки до виконання курсового проекту «Теорія автоматичного управління», для здобувачів спеціальності 174 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка» освітньо-професійних програм «Комп’ютерно-інтегровані системи автоматики та робототехніки»,

«Комп’ютеризовані системи управління та прикладне програмування». / укл. І.Р. Козбур, В.П. Пісьціо, В.Б. Савків, П.С. Федорів. – Тернопіль: ТНТУ імені Івана Пулюя, 2025. – 67 с

30. Методичні вказівки для написання розділу «Безпека життєдіяльності, основи охорони праці» в кваліфікаційних роботах здобувачів освітнього рівня „бакалавр”. Для студентів всіх форм навчання рівень вищої освіти перший (бакалаврський)/ укл.: О. Я. Гурик , І. Б. Окіпний. – Тернопіль: ТНТУ імені Івана Пулюя, 2021. - 20 с.
31. Навчально-методичний посібник до практичних заняття з дисципліни «Безпека життєдіяльності, основи охорони праці» для студентів освітнього ступеня бакалавр усіх спеціальностей та форм навчання / Укладачі : О. Я. Гурик, І. Б. Окіпний, В. С. Сенчишин, С. Ю. Мариненко, О. І. Король. Тернопіль : ТНТУ імені Івана Пулюя, 2025. 123 с. https://elartu.tntu.edu.ua/handle/lib/48496
32. Безпека в надзвичайних ситуаціях. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання / укл.: Стручок В. С. Тернопіль: ФОП Паляниця В. А., 2022. 156 с.
33. Савків В. Б., Ковтко А. М. Мутаційне тестування як механізм зворотного зв’язку в автоматизованій системі генерації модульних тестів на основі великих мовних моделей. Електротехнічні та інформаційні системи. 2026. № 109. С. 278–287. DOI: https://doi.org/10.32782/EIS/2026-109-33.
34. Ковтко А., Савків В. Архітектура автоматизованої системи генерації тестів. Herald of Khmelnytskyi National University. Technical Sciences. 2026. Т. 365, № 3. С. 346–352. DOI: https://doi.org/10.31891/2307-5732-2026-365-48.
35. Ковтко А.М., Козбур І.Р., Савків В.Б., Козбур Г.В. Архітектура автоматизованої системи для генерації модульних тестів на основі штучного інтелекту та мутаційного тестування // Актуальні задачі сучасних технологій : зб. тез доповідей XIV міжнар. наук.-техн. конф. молодих учених та студентів, (Тернопіль, 11-12 грудня 2025) / М-во освіти і науки України,

Терн. націон. техн. ун-т ім. І. Пулюя [та ін]. – Тернопіль : ФОП Паляниця В.А., 2025 – с. 278–280.

36. Ковтко А. М., Савків В. Б., Козбур І. Р. Автоматизована системи генерації модульних тестів на основі штучного інтелекту та мутаційного тестування // Тези XIII МНПК „Актуальні задачі сучасних технологій“, Тернопіль, 11-12 грудня 2024 року. 2024. С. 417–418. http://elartu.tntu.edu.ua/handle/lib/47912
37. Kovtko A., Savkiv V., Kozbur H., Kozbur I., Trembach R. Integration of mutation testing into unit test generation using large language models // Proceedings of the 3rd International Workshop on Computer Information Technologies in Industry 4.0 (CITI’2025). Ternopil, Ukraine, June 11–12, 2025. Vol. 4057. CEUR Workshop Proceedings, 2025. https://ceur-ws.org/Vol-4057/paper4.pdf
38. Микитишин А. Г., Митник М. М., Стухляк П. Д., Пасічник В. В. Комп’ютерні мережі. Книга 1 [навчальний посібник]. Львів : «Магнолія 2006», 2013. 256 с.
39. Микитишин А. Г., Митник М. М., Стухляк П. Д., Пасічник В. В. Комп’ютерні мережі. Книга 2. [навчальний посібник]. Львів : "Магнолія 2006", 2014. 312 с.
40. Микитишин А. Г., Митник М. М., Стухляк П. Д. Телекомунікаційні системи та мережі. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2017. 384 с.
41. Буров Є., Митник М. Комп’ютерні мережі. (у 2-х томах). Львів, Магнолія, 2018.
42. Комплексна безпека інформаційних мережевих систем. Навчальний посібник для студентів спеціальності 174 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка» / А. Г. Микитиший, М. М. Митник, О. С. Голотенко, В. В. Карташов. – Тернопіль: ФОП Паляниця В.А., 2023. – 324 с. https://elartu.tntu.edu.ua/handle/lib/42625
43. Конспект лекцій з дисципліни «Комп’ютерні системи штучного інтелекту» для студентів спеціальності G7 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка» денної та заочної форми здобуття освіти / уклад. І. С. Дідич. // ТНТУ. – 2025. – С. 92. https://elartu.tntu.edu.ua/handle/lib/50293

44. Пилипець М. І. Правила заповнення основних форм технологічних документів : навч.-метод. посіб. / Уклад. Пилипець М. І., Ткаченко І. Г., Левкович М. Г., Васильків В. В., Радик Д. Л. Тернопіль : ТДТУ, 2009. 108 с. https://elartu.tntu.edu.ua/handle/lib/42995

45. Паламар М. І., Стрембіцький М. О., Паламар А. М. Проектування комп'ютеризованих вимірювальних систем і комплексів : навч. посіб. Тернопіль : ТНТУ, 2019. 150 с.
46. Савків В.Б., Капаціла Ю.Б., Михайлишин Р.І. Методичні вказівки до виконання кваліфікаційної роботи бакалавра спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології». Тернопіль.: Видавництво ТНТУ. 2021. 50 с. https://elartu.tntu.edu.ua/handle/lib/35172