Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Перший (бакалаврський)
(рівень вищої освіти)

Розроблення програмного забезпечення для аналізу вхідної інформації робітника приладобудівного виробництва для видачі завдань на виконання
(тема)

Виконав:
здобувач _4_ року навчання,
групи АКТСІ-21-2
Антон АНДРЕЄВ
(власне ім'я, прізвище)
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Системна інженерія
(повна назва освітньої програми)
Керівник доц. Артем БРОННІКОВ
(посада, власне ім'я, прізвище)

Допускається до захисту
Зав. кафедри КІТАР _____ Ігор НЕВЛЮДОВ
(підпис) (влаасне ім'я, прізвищу)

2025 р.

Я, Андреєв Антон Сергійович, як здобувач вищої освіти ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Я не використовував штучний інтелект для підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на віповідне джерело.

"04" червня 2025 р.  Антон АНДРЕЄВ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет Автоматики та комп'ютеризованих технологій
Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
Рівень вищої освіти Перший (бакалаврський)
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
Тип програми Освітньо-професійна
Освітня програма Системна інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«28» квітня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студентові Андреєву Антону Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення програмного забезпечення для аналізу вхідної інформації робітника приладобудівного виробництва для видачі завдань на виконання
Затверджена наказом по університету від 19.05 2025 р. № 391 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 11.06.2025
3. Вихідні дані до роботи бібліотека для використання openai моделей, машинне навчання, LLM

4. Перелік питань, що потрібно опрацювати в роботі
4.1 Вступ
4.2 Аналіз літератури за темою
4.3 Вибір та обґрунтування технічних засобів для створення програмного забезпечення
4.4 Розроблення програмного забезпечення

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Графічний матеріал у вигляді презентації (10 с.)

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Приміт-ка
1	Аналіз технічного завдання	28.04-04.05.2025	виконано
2	Опрацювання літератури за темою роботи	05.05-10.05.2025	виконано
3	Аналіз літератури за темою	11.05-17.05.2025	виконано
4	Вибір і обґрунтування технічних засобів для створення програмного забезпечення	18.05-25.05.2025	виконано
5	Розроблення програмного забезпечення	26.05-31.05.2025	виконано
6	Подання роботи на перевірку Інтернет-сервісом StrikePlagiarism	01.06-03.06.2025	виконано
7	Оформлення пояснювальної записки	04.06-08.06.2025	виконано
8	Подання роботи на рецензію	09.06-11.06.2025	виконано
9	Подання роботи на підпис зав. кафедри	12.06-14.06.2025	виконано
11	Подання кваліфікаційної роботи в ЕК	15.06.2025	виконано

Дата видачі завдання 28.04.2025 р.

Здобувач _____ Антон АНДРЕЄВ
(підпис) (власне ім'я, прізвище)

Керівник роботи _____ доц. Артем БРОННІКОВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка: 93 с., 4 табл., 24 рис., 3 дод., 31 джерело.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, АНАЛІЗ ДАНИХ, АВТОМАТИЗАЦІЯ, УПРАВЛІННЯ ПІДПРИЄМСТВОМ, ШТУЧНИЙ ІНТЕЛЕКТ.

Об’єкт розробки – автоматизація роботи приладобудівного виробництва.

Предмет розробки – аналіз вхідної інформації робітника для видачі завдань на виконання.

Мета роботи – розробка програмного забезпечення для автоматизованого аналізу вхідних даних приладобудівного виробництва.

В роботі розглянуто актуальні питання за темою, запропоновані рішення з аналізу даних на сучасному виробництві, розроблено програмне забезпечення для автоматизованого аналізу вхідних даних приладобудівного виробництва.

ABSTRACT

Explanatory note: 93 pages, 4 tabl., 24 fig., 3 app., 31 sources.

SOFTWARE, DATA ANALYSIS, AUTOMATIZATION, MANUFACTURE MANAGEMENT, ARTIFICIAL INTELLIGENCE.

The object of development is автоматизація роботи приладобудівного виробництва.

The subject of development is the analysis of employee input information to assign tasks.

The purpose of the work is developing software for automatized input data analysis of electronic manufacturer.

In the work, the actual issues in the topic have been considered, modern solutions in data analysis in manufacturing have been proposed, and software has been developed to automate data analysis of electronic manufacturer.

ЗМІСТ

Перелік умовних скорочень 6

Вступ 7

1 Аналіз літератури за темою 9

1.1 Керування виробництвом 9

1.2 Аналіз використання штучного інтелекту для аналізу даних 13

1.3 Використання LLM в аналізі даних 18

1.4 Використання API для опрацювання даних 23

2 Вибір і обґрунтування технічних засобів для створення програмного забезпечення 27

2.1 Вибір мови програмування та технічних засобів 27

2.2 Схема роботи програмного забезпечення. 36

2.3 Прості запити до системи та аналіз об'єктів у файлі 38

2.4 Розрахунки 42

3 Розроблення програмного забезпечення 45

3.1 Блок-схема роботи програми 45

3.2 Розроблення додатку 47

3.3 Розроблення інтерфейсу 90

3.4 Охорона праці 93

Висновки 96

Перелік джерел посилання 98

Додаток А Апробація результатів роботи 102

Додаток Б Код програми 105

Додаток В Демонстраційний матеріал 170

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – програмне забезпечення;

ШІ – штучний інтелект;

AI – Artificial intelligence;

API – Application program interface;

CEO – Chief executive officer;

CFO – Chief financial officer;

CTO – Chief technical officer;

CVO – Chief visionary officer;

HTTP – Hypertext transfer protocol;

IP – Internet protocol;

JSON – JavaScript object notation;

LLM – Large language model;

PLC – Programmable logic controller;

TCP – Transfer control protocol.

ВСТУП

На сьогоднішній день, необхідність автоматизації підприємств зростає. Підприємства збільшують все частіше використовують штучний інтелект для аналізу даних, в тому числі і великі мовні моделі, здатні перетворювати натуральний людський текст. За допомогою цих інструментів можна швидко та дешево запевнити початковий аналіз даних, перетворення їх до форми, яка більше підходить до зберігання в базі даних підприємства та зменшити обсяг роботи людини, який потрібен для аналізу. Крім того це дає можливість зменшити вплив людини на підприємстві та вкладати більше кошів у якість продукції через зменшену кількість робочих місць. Разом з необхідністю автоматизації підприємства зростає потреба у вдосконаленні уже існуючого програмного забезпечення для аналізу даних та створення нових, більш сучасних інтелектуальних систем для аналізу даних.

Об’єкт розробки – автоматизація роботи приладобудівного виробництва.

Предмет розробки – аналіз вхідної інформації робітника для видачі завдань на виконання.

Метою кваліфікаційної роботи є розробка програмного забезпечення для автоматизованого аналізу вхідних даних приладобудівного виробництва.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести аналіз технологій штучного інтелекту та аналізу даних;

– розробити структурну схему макету;

– провести підбір елементної бази;

– оформити кваліфікаційну роботу згідно та ДСТУ 3008:2015 [1], а також з підручником та методичними вказівками з підготовки й оформлення кваліфікаційної роботи [2-3].

Дана робота пройшла апробацію у журналі ADED-2025 (1) [4]. Проведені дослідження відповідають цілям сталого розвитку (ЦСР), а саме:

ЦСР 4, ЦСР 7, ЦСР 9, ЦСР 11, та ЦСР 12.

1 АНАЛІЗ ЛІТЕРАТУРИ ЗА ТЕМОЮ

1.1 Керування виробництвом

Виробництво – дуже складна галузь діяльності. Створення матеріальних (та не тільки) благ в індустріальному масштабі вимагає координації багатої кількості працівників та машин, та наймаліші порушення такої координації можуть призвести до катастрофічних наслідків. Саме тому дуже важливим є керування цим процесом, як ручне так і автоматизоване.

Програмне забезпечення для аналізу вхідних даних робітників швидко формує виробничі завдання й вирівнює план-факт, штучний інтелект і машинне навчання додають гнучкості системам керування та дають змогу проактивно реагувати на коливання попиту, семантичні мережі узгоджують знання між підрозділами й зменшують простої, web-додатки для диспетчеризації відкривають доступ до показників лінії з будь-якого пристрою, PHP і MySQL забезпечують модульним MES-системам швидку інтеграцію з наявними базами даних, QR-коди прискорюють ідентифікацію партій і мінімізують помилки пакування, гейміфікаційні платформи скорочують час підготовки операторів устаткування, ефективний пошук технологічної інформації в інтернеті дозволяє оперативно оновлювати виробничі інструкції, порівняння робототехнічних платформ допомагає обрати базу для впровадження колаборативних роботів на дільниці, поєднання комп’ютерних ігор із веб-дизайном наочно відображає KPI ліній у реальному часі, а аналіз симуляторів Webots, CoppeliaSim та Gazebo окреслює їхню роль у створенні цифрових двійників для планування й удосконалення виробничих процесів [5-14].

Історично керуванням виробництва займався власник засобів виробництва. Така особа, розуміючи, що не може адекватно обслуговувати велику кількість машин, наймає працівників, які за фіксовану плату займаються обслугою засобів виробництва. Виробництво та адекватність

виконаної працівником праці контролює сам власник бізнесу, що може працювати за умовою відносно низької кількості робочої сили, але велика кількість працівників робить таку схему організації праці неможливою.

Тому з’явився клас менеджерів середньої ланки – вони все ще є найманими працівниками, але при цьому також беруть на себе частину керування виробництвом, наприклад, керують однією з невеличких команд (рис. 1.1).

Рисунок 1.1 – Прикладна схема контроля великого виробництва (заводу)

Але що робити, якщо власників бізнесу багато? В такому разі ефективно розподілити та керувати командою працівників стає неможливо. В такому разі призначають Головного виконавчого директора (англ. CEO). Цей менеджер може (але не мусить) бути одним з власників бізнесу, але може бути теж найманим працівником, який отримує фіксовану зарплату.

Така система, коли кожний працівник має окремого начальника, називається лінійною. Вона має багато переваг: по-перше, її використання значно покращує дисципліну команди, гарантуючи субординацію працівників. До того ж, значно легше стає приймати й запроваджувати рішення, які стосуються виробничого процесу.

На жаль, така система управління має теж свої вади. По-перше, структура таких виробництв значно зменшує можливість працівників впливати на рішення вищого менеджменту. По-друге, через негнучкість системи впровадження інновацій часто стає майже неможливим. По-третє, в

великих підприємствах такі структури часто безконтрольно розростаються, призводячи до значної витрати коштів.

Альтернативою такій системі може служити функціональна структура управління (рис. 1.2). В такій структурі працівники не є приписані до конкретного начальника, але належать до функціональних підрозділів, які зазвичай створюються для вирішення окремого завдання.

Рисунок 1.2 – Функціональна схема управління

Функціональна система управління теж має свої вади. Головним образом вони виводяться з того, що стає набагато складніше слідкувати та збирати дані підприємства. Також проблемою є порушення принципу єдиного керівництва: можливість працівника мати більше ніж одного начальника може призвести до дуже нерівномірного розподілу праці.

Система, яка комбінує ознаки лінійної та функціональної системи називається лінійно-функціональною. Найчастіше на виробництві стосується саме така система. Така система дозволяє поєднати дисципліну та організованість лінійної системи з гнучкістю та швидкістю функціональності (рис. 1.3).

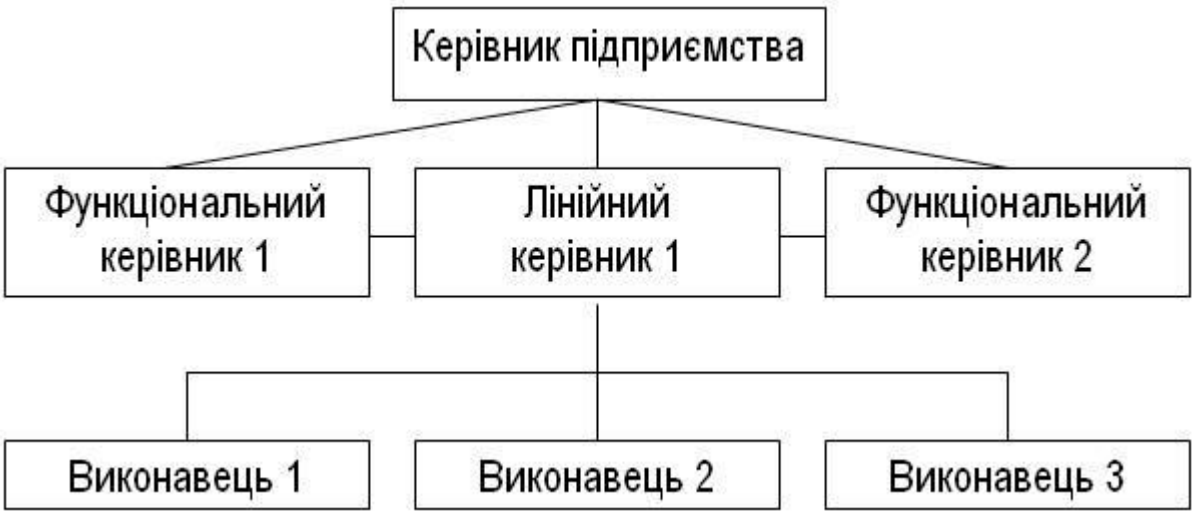

Рисунок 1.3 – Лінійно-функціональна система управління

Посада генерального директора не є єдиною вищою посадою у фірмі. Часто керівництво відбувається за участі декількох директорів, кожен з яких відповідає за окрему функцію підприємства. Деякі найбільш поширені ролі включають в себе.

Технічного директора (CTO): технічний директор займається в першу чергу технічними справами, керуючи технічним підрозділом фірми. Наприклад, на підприємстві, яке займається інженерією програмного забезпечення, CTO буде відповідати за координацію праці різних команд програмістів.

Віце-директора (CVO): зазвичай ця посада є заступником директора. Заступник має обов’язок підтримувати й заміняти директора, коли він, наприклад, недоступний.

Фінансового директора: ця посада обіймає управління фінансами фірми. Фінансові директори визначають бюджет, фінансовий напрям фірми та роблять різного роду фінансові підрахунки.

Останніми часами все більше розмов йде про автоматизацію позиції менеджера. Розвиток комп’ютерних технологій робить традиційне управління людьми все менш ефективним. Завдяки розвитку штучного інтелекту можна легко автоматизувати нагляд та організацію праці команди, що значно зменшує навантаження на начальників та значно збільшує розмір їх

підрозділів. Це усуває необхідність у розгалуженій системі менеджменту та допомагає заощадити багато фінансів.

1.2 Аналіз використання штучного інтелекту для аналізу даних

В сучасному світі, все більше виробництв переходить з ручної, мануальної роботи на автоматизацію. Нові технології в сфері машинобудування та комп'ютеризації дозволяють в багато разів збільшити продуктивність працівника.

Автоматизація – це застосування наукових досягнень в сфері математики, теорії управління, автоматики, комп’ютерних та інших наук з метою якнайбільше зменшити потребу в участі людини в виробничому процесі.

Дуже сильно на автоматизацію праці вплинули PLC – Programmable logic controllers, індустріальні та дуже довготривалі комп’ютери, спеціально створені для автоматизації великих виробництв. PLC та мікроконтролери дозволяють творити комплексні системи, які можуть значно полегшити управління виробництвом.

Наприклад, на виробництві, яке спеціалізується на виробництві палива, PLC може бути підключений як до машин, які проводять перегонку нафти, так до датчиків, які можуть перевірити вихідний продукт на якість, а також до протипожежних датчиків, після сигналу з яких можна швидко зупинити роботу виробництва, запобігаючи можливим травмам серед персоналу та фінансовим втратам.

Ще один спосіб автоматизувати виробництво – використання робототехніки. Роботи допомагають замінити ручну працю, яка заснована на монотонних діях, які не потребують мислених зусиль. Це може сильно зменшити кошти виробництва (не треба утримувати такий великий штат працівників) та значно зменшити кількість травматичних випадків на виробництві (рис. 1.4).

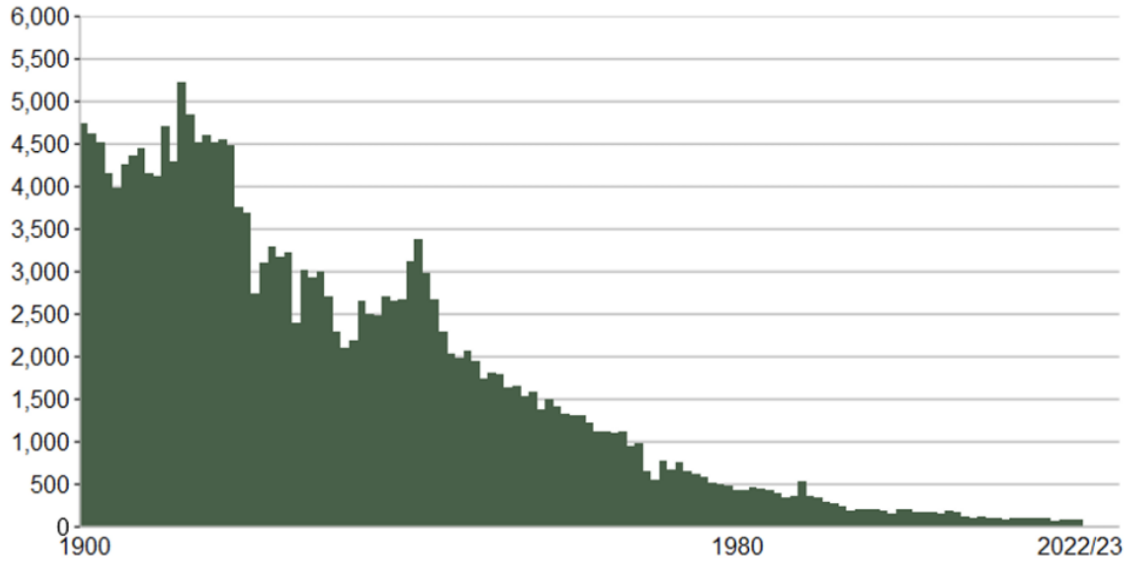

Рисунок 1.4 – Зменшення кількості травм на виробництві можна пояснити автоматизацією

Штучний інтелект – це здатність комп'ютера виконувати завдання типові для людського мозку, такі як навчання, аналітичне мислення, розпізнавання інформації та прийняття рішень. Штучний інтелект має свої застосування в низці областей комп'ютерних наук. Як приклад можна привести технології комп'ютерного зору та пошук інформації.

Комп'ютерний зір (автоматичне розпізнавання інформації на підставі графічних зображень) не можна уявити без використання ШІ. Зазвичай для цього використовуються нейромережі – системи, змодельовані на підставі мізків живих істот. Найчастіше таку мережу спочатку "навчають" – автоматично змінюють ваги різних нейронів на підставі зображень та їх класифікації (даних для тренування). Наприклад, компанії, які виробляють автомобілі з автопілотом, використовують технології комп'ютерного зору для розрізнення об'єктів на дорозі.

Google, одна з найбільших AI-корпорацій, була заснована саме як компанія, яка спеціалізувалася на пошуку інформації в інтернеті. Сучасні технології штучного інтелекту дозволяють значно покращити якість та вірогідність пошуку. Наприклад, аналіз сторінки при індексації може

допомогти відфільтрувати шкідливу або протиправну інформацію. Інше можливе застосування штучного інтелекту – можливість швидко підсумувати зміст веб-сторінки.

Штучний інтелект (і, особливо, машинне навчання) нерозривно пов'язаний з аналізом даних. Підприємства, особливо ті, які пов’язані з обробкою великої кількості даних (наприклад, банки та фінансові аналітики) широко використовують такі засоби.

Як приклад можна навести експертні системи. Експертна система – це комп’ютерна система, яка емулює процес прийняття рішення людиною-експертом. Перші експертні системи почали з’являтися ще в 70-х роках 20 сторіччя, коли виникла потреба автоматизувати окремі аспекти експертної роботи на підприємствах. Одною з перших таких систем була MYCIN, яка повинна була допомогти в ідентифікації бактеріальних інфекцій.

Експертні системи зазвичай складаються з двох частин: бази знань та машини висновування. База знань (найчастіше, база даних) містить в собі набір речень, які представляють собою знані факти. Машини висновування, в свою чергу, є логічним модулем експертної системи. Вони можуть застосовувати набір логічних правил для того, щоб виводити нові факти з існуючої бази знань. Найчастішою архітектурою машини висновування є архітектура базована на структурах правил типу IF -> THEN, що дозволяло досить ефективно для свого часу виводити висновки з наборів знань.

Якщо в висновках виникають конфлікти, їх вирішенням займається інтерпретатор правил. Коли машина висновування ідентифікує кілька можливих правил, які можна було б застосувати до бази знань, інтерпретатор, циклічно проходячи по всім правилам, дозволяє ідентифікувати ті, які були б найбільш корисні в даному контексті. Алгоритм вибору правил можна коригувати ззовні за потреби користувача експертної системи (рис. 1.5). Після того, як відповідне правило було відібране, експертна система виконує дію, передбачену правилом, наприклад запис даних до файлу чи бази даних,

змінення бази знань чи правил, або інша дія (наприклад, форматування та подання інформації користувачу).

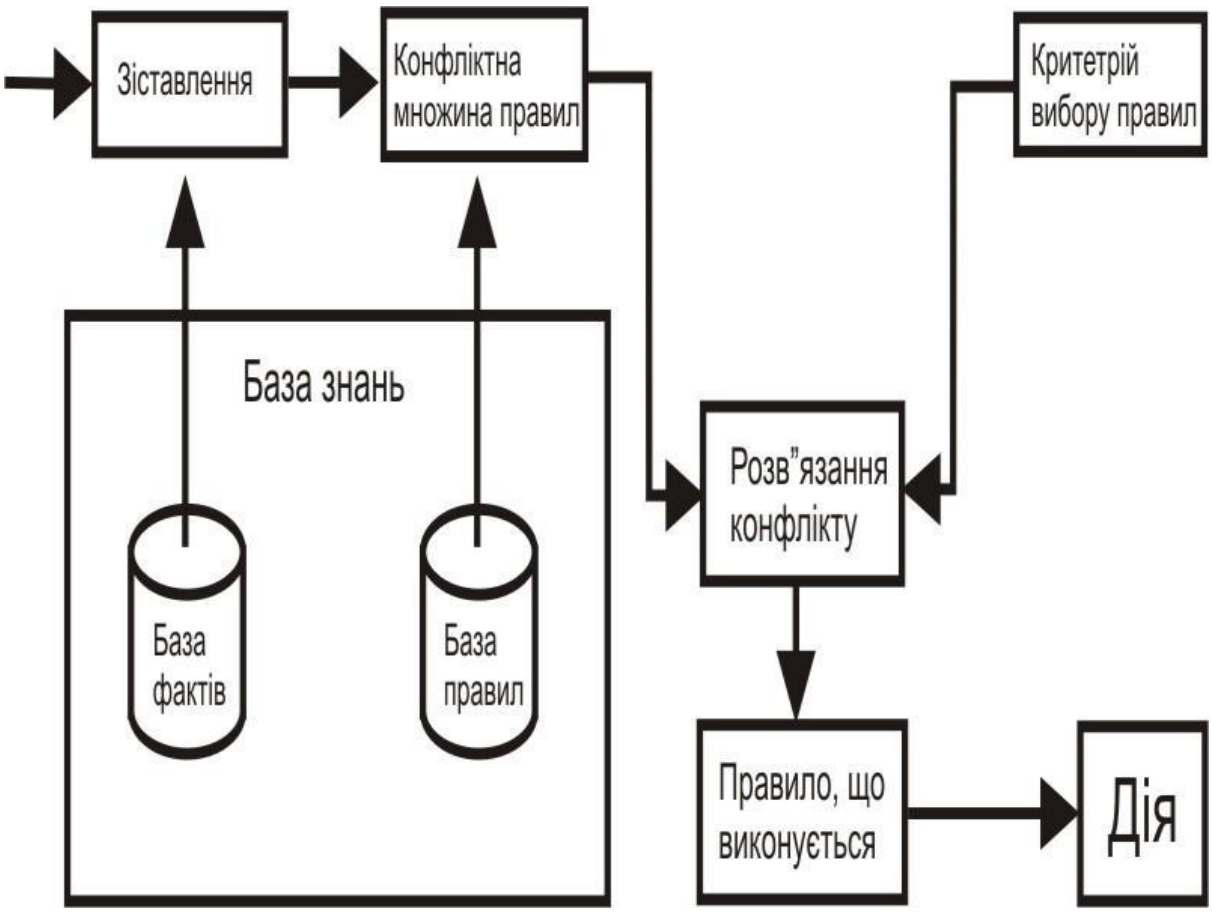

Рисунок 1.5 – Схема діяльності експертної системи

Сфери використання експертних систем дуже обширні та стосуються всіх областей аналізу даних. Наприклад, експертні системи в медицині можуть значно полегшити та прискорити діагностику захворювань, приймаючи інформацію показників та симптомів пацієнта та виводячи список можливих хвороб, які можуть в нього бути.

Ще одна дуже часта сфера використання експертних систем – фінансовий аналіз. Маючи обширну базу знань фінансових операцій та достатньо правил їх аналізу, такі системи можуть дуже точно передбачити ситуацію на ринку та дати рекомендації, наприклад, щодо інвестицій.

На виробництві експертні системи частіше всього використовуються, щоб спостерігати за показниками датчиків. Наприклад, PLC, на якому може

бути встановлена така система, має здатність прочитати показники, порівняти їх з нормою та в залежності від результатів такого порівняння прийняти рішення, що треба зробити (змінити показник, включити якийсь механізм, повідомити персонал про несправність тощо).

Плюси експертних систем – порівняльно низькі кошти оперування, прозорий механізм прийняття рішень (користувач може легко дізнатися, чому саме був зроблений той чи інший висновок), швидкість та надійність роботи. Ще один великий плюс – порівняльно нижча потреба в засобах комп'ютера, ніж в інших типів штучного інтелекту, що дозволяє запускати експертну систему навіть на досить непродуктивних комп'ютерах (наприклад, PLC).

Мінуси таких систем – велика база знань може зробити опрацювання досить простих запитів порівняльно дорогим в обчислюванні. Також проблеми виникають з отриманням бази знань: дані до неї повинна вводити низка експертів, бо часто необхідних даних може просто не бути в вільному доступі. Ще одна велика проблема – експертна система не може сама встановлювати глибокі концепції та відносини між фактами та потребує для цього участі експерта.

Також частим місцем застосування машинного навчання є сфера охорони здоров'я: для виявлення та діагностики раку використовують моделі, побудовані на зібраних даних пацієнтів. Ще одна дуже поширена галузь застосування ШІ – лікування: штучний інтелект може допомогти в проведенні хірургічних операцій, полегшуючи планування та поставляючи хірургу необхідну інформацію щодо ходу процедури, знижуючи ризик перенавантаження (рис. 1.6) [15].

Рисунок 1.6 – Використання AI в дослідженнях раку

Як можна побачити, штучний інтелект обіймає дуже широкий круг проблем, для вирішення яких раніше необхідний був вклад людини. Завдяки здатності ШІ аналізувати дані та знаходити закономірності можна автоматизувати речі, для виконання яких раніше був потрібний штаб досвідчених експертів.

1.3 Використання LLM в аналізі даних

Останнім часом популярність набувають LLM – Large Language Models. LLM – це клас алгоритмів штучного інтелекту, розроблених для обробки

натурального тексту (natural language processing) та тренованих з допомогою самокерованого навчання на великих об’ємах даних.

Для навчання LLM використовують трансформери – архітектуру, яка дозволяє токенізувати текст, виділяючи з нього більш і менш важливі елементи, що значно спрощує аналіз тексту та дозволяє витрачати менше часу на навчання такої моделі.

Для цього в трансформерах стосується механізм уваги – метод, який приділяє ваги до кожного токена (в випадку великих мовних моделей це окремі слова), та визначає вектори, які описують взаємну дистанцію між токенами. Ці ваги, на відміну від класичних, можуть постійно змінюватися, що забезпечує більшу гнучкість під час навчання та предикції (рис. 1.7).

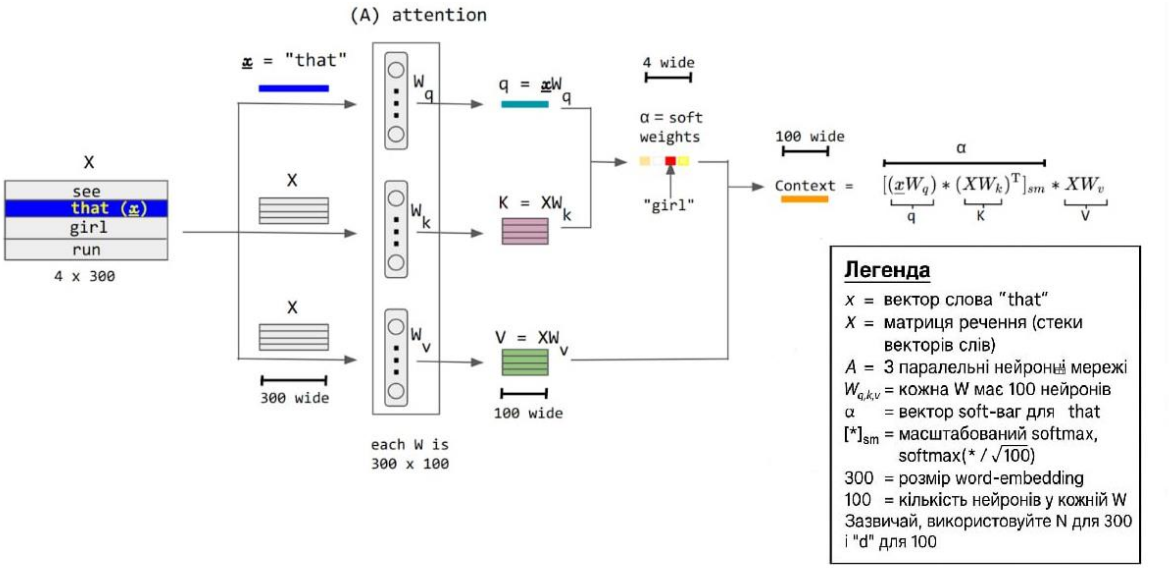

Рисунок 1.7 – Схема, яка ілюструє механізм уваги

Результатом перебігу такої LLM є предикція – передбачення наступних слів на підставі контексту. З цього виходить, що чим більший розмір контексту та обсяг даних до навчання, тим більш детальною буде предикція.

В табл. 1.1 наведено порівняння контексту популярних великих мовних моделей.

Таблиця 1.1 – Порівняння контексту популярних великих мовних моделей

Модель	Розмір контексту (в токенах)
GPT-3	2048
GPT-4	8192 (може бути іншим в різних конфігураціях)
Llama 3.1 8B	128,000
DeepSeek-R1-Distill-Llama-70B	128,000

LLM вигідно виділяються на тлі своїх попередників, бо не потребують контрольованого навчання, що дозволяє використовувати набагато більшу кількість даних. З цього також випливає найбільший мінус такої моделі: чим більше датасет, тим більша ефективність моделі, що робить розробку LLM значно дорожчою.

Сучасні підприємства дуже часто та глибоко інтегрують LLM в свої виробництва.

Завдяки своїм майже ідеальним здатностями обробки натурального тексту, такі моделі можуть виконувати дуже різні завдання в багатьох сферах. З кінця 2022 року (коли була представлена перший популярний інтерфейс для LLM, ChatGPT) кількість користувачів досягла більше ніж 180,5 мільйонів користувачів.

Можливі сфери використання LLM на підприємствах:

– автоматизація підтримки користувачів;
– полегшення комунікації серед працівників;
– опрацювання внутрішніх документів організації;
– планування;
– полегшення розробки та інженерії програмного забезпечення;
– аналіз вхідних даних.

Статистика використання ШІ наведено на рис. 1.8.

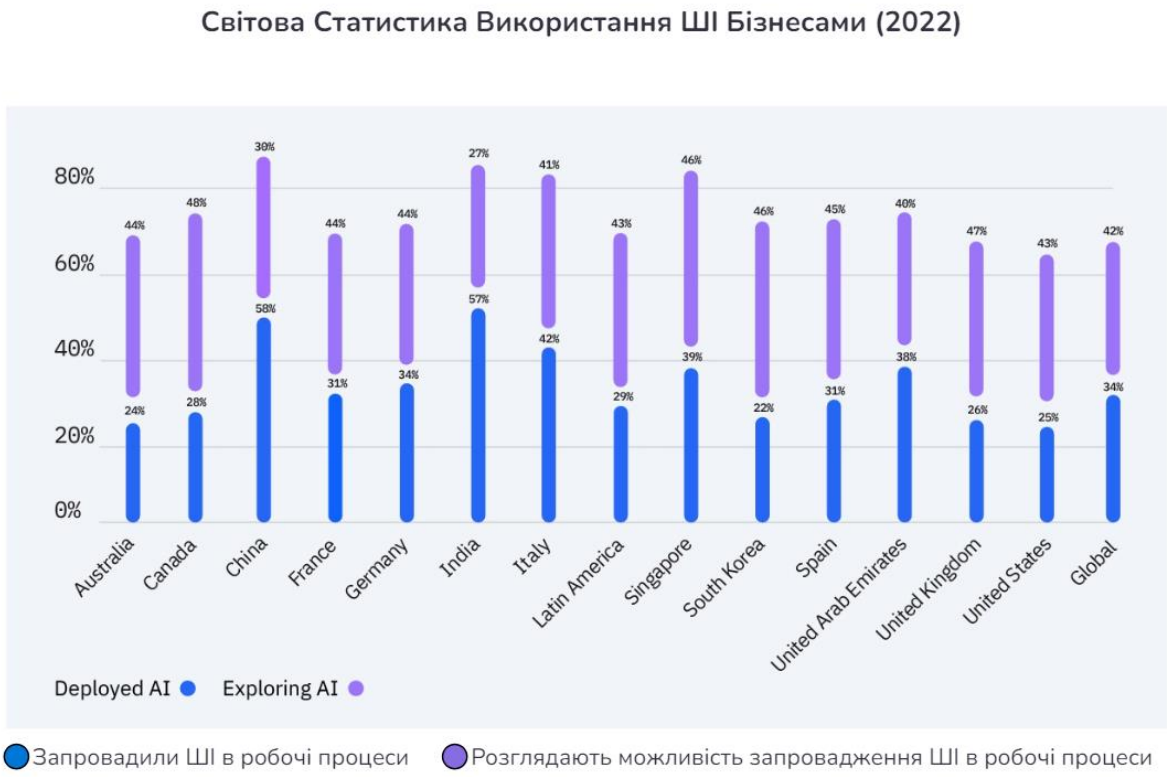

Рисунок 1.8 – Статистика використання ШІ бізнесами [16]

Автоматизація підтримки користувача може бути виконана за допомогою ШІ-асистента, який допомагав би користувачу дізнатися більше інформації щодо компанії та продуктів, які вона пропонує, без витрачання коштів на професійного асистента. Це значно пришвидшує комунікацію між користувачем та компанією в тривіальних випадках, а в разі питань, які потребують участі людини, LLM може підключити користувача до асистента. Мінусом такого підходу є те, що LLM часто “галюцінують” та можуть видати вкрай невірну інформацію, що може зашкодити іміджу компанії та відштовхнути користувача.

Полегшення комунікації між працівниками в основному полягає на тому, що працівники можуть скоротити якийсь текст (наприклад, транскрипцію наради) та витрачати менше часу на то, щоб проаналізувати, наприклад, висновки наради. Також LLM можуть значно полегшити процес

написання повідомлень електронної пошти, що по-перше, значно прискорює цей процес, а по-друге, може допомогти, якщо працівник, наприклад, не володіє мовою спілкування на достатньому рівні та не впевнений, що десь не допустив помилку. Цей аспект використання LLM, попри свою, видавалося б, користь, може бути вкрай руйнівним при неправильному використанні, бо скорочення тексту не завжди будуть містити повну інформацію (а інколи, навпаки, можуть “згалюцінувати” невірні факти).

Опрацювання та аналіз внутрішніх документів за допомогою LLM полягає на тому, що велика мовна модель приймає як вхідні дані документ (будь-який, наприклад, фінансові дані) та видає результат аналізу з певної точки зору (наприклад, бюджетну рекомендацію). Деякі компанії, які займаються розробкою LLM, дозволяють настроїти її так, щоб вихідні дані могли бути легко оброблені комп’ютером (наприклад, в форматі JSON). Мінус такого підходу – те, що він все одно потребує участі людини, яка буде перевіряти, чи правильно був виконаний аналіз.

Планування може бути полегшено чи взагалі автоматизовано, використовуючи обширні тренувальні дані великих мовних моделей, які можуть містити в собі подібні проекти до того, яким займається компанія, та може дати поради щодо планування. Дуже великим мінусом такого підходу є те, що не завжди такі поради будуть корисні (а інколи – вкрай шкідливі).

Автоматизація розробки ПЗ – одна з найбільш поширених сфер застосування LLM. Через дуже велику кількість інформації щодо розробки ПЗ, яка знаходиться в вільному доступі, багато задач, які потребували би молодшого спеціаліста з розробки ПЗ, можуть бути виконані автоматично. Це можна побачити, наприклад, по змінах на ринку праці розробників ПЗ: більша частина роботи, яку виконували програмісти з невеликим опитом, виконується LLM, тому все менше і менше підприємств пошукують таких програмістів. В такому підході є кілька дуже великих мінусів: по-перше, часто великі мовні моделі містять помилки в тренувальних даних, що може вкрай негативно вплинути на працю ПЗ (або програмний код може навіть не запрацювати). По-

друге, ШІ частіше за все не може передбачити всі луки в безпеці програми (особливо в кібербезпеці), через що часто на виправлення таких програм витрачається більше часу, ніж було б треба, якщо б програму писав молодший спеціаліст.

Використання LLM до аналізу вхідних даних – дуже поширене явище, яке часто зустрічається в фірмах, де таких даних дуже багато. Наприклад, платформи для обміну повідомленнями, які зобов'язані запобігати поширенню нелегальної інформації, можуть значно пришвидшити свою роботу завдяки перегляду повідомлень, які ШІ визнає за підозрілі. Також великі мовні моделі часто використовуються під час пошуку працівників: в випадку великої кількості кандидатів вони можуть дозволити вибрати тільки тих, які найкраще підходять до своєї ролі. Мінусом такого застосування є непевність даних, які видає велика мовна модель, але в випадку великої кількості вхідних даних такою непевністю можна знехтувати.

1.4 Використання API для опрацювання даних

Абсолютна більшість даних сьогодні передається через мережу Інтернет. Саме тому дуже важливо звернути увагу на те, яким саме чином проходить ця передача. Основним елементом комунікації в мережі є сокет – інтерфейс операційної системи, через який можна пересилати дані між двома комп'ютерами або навіть між двома процесами на тому самому комп'ютері (що є основою для мікросервісних технологій, де якась задача розбивається на кілька дрібних задач, які потім виконуються різними програмами).

В Інтернеті використовується протоколи TCP та IP. Сокети, які використовують TCP-IP, називаються теж інтернет-сокетами. Transmission Control Protocol (TCP) – це протокол, який робить можливим підключення та трансмісію даних з одного пристрою до іншого. Основна ідея TCP – забезпечення надійності та цілісності даних, які отримує клієнт (рис. 1.9).

Для цього TCP має низку різноманітних елементів, які значно сповільнюють та значно збільшують обсяг даних, але завдяки дуже високій редундантності допомагають зберегти дані цілісними.

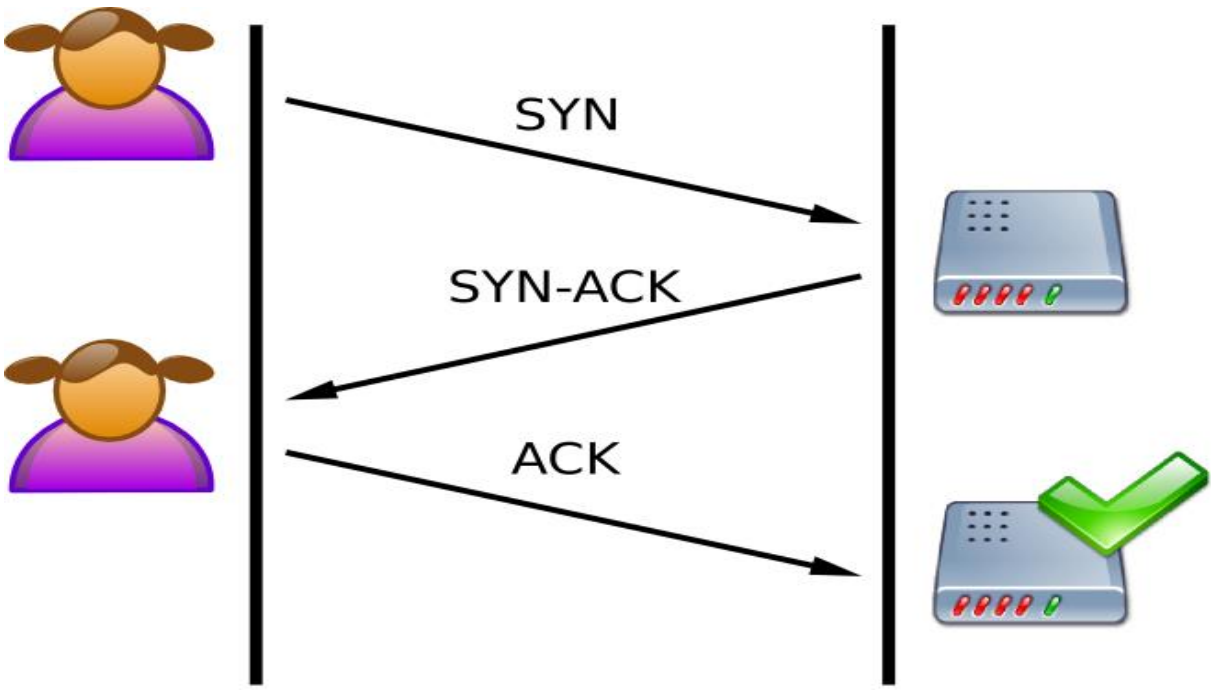

Рисунок. 1.9 – Схема встановлення з’єднання в TCP/IP

Зазвичай, сесія обміну даними з TCP виглядає так:

– фаза підключення (handshake) – це коли клієнт відправляє до сервера запитання, яке містить номер порту клієнта, номер порту сервера та інші метадані (крок SYN). У відповідь, сервер відповідає клієнту (SYN-ACK). Важливо, що обмін даними не починається, поки клієнт не підтвердить, що отримав відповідь (ACK);

– передача даних – це дані передаються клієнту від сервера у вигляді пакетів. Якщо пакет був пошкоджений під час передачі (або взагалі не дійшов до клієнта), він вважається втраченим та сервер буде повторювати спробу передати його, поки клієнт не підтвердить, що всі пакети були прийняті;

– роз’єднання – це коли один з учасників обміну даними (неважливо, клієнт чи сервер) ініціює закінчення з’єднання, висилаючи до іншого

запитання FIN, й переходить у стан очікування на закінчення з’єднання (FIN-WAIT-1). Другий учасник отримує інформацію та посилає запитання ACK, після отримання якого перший переходить в стан FIN-WAIT-2. Після обробки всіх запитань другий учасник посилає запитання FIN до першого. Перший відповідає запитанням ACK та закінчує роботу, натомість другий приймає це запитання й лише після цього закінчує роботу.

HTTP (Hypertext transfer protocol) – це протокол передачі даних, заснований на TCP-IP, який передбачає відправку структурованих повідомлень з клієнта до сервера. Окрім інформації, яку треба переказати, такий протокол зазвичай містить в собі інформацію щодо конкретної веб-сторінки сайту, на який подається посилання, метадані, які описують, наприклад, браузер користувача, а також тип запити.

Можливі запити HTTP наведені в табл. 1.2.

Таблиця 1.2 – Можливі запити HTTP

Тип запиту	Ціль існування
GET	Отримання інформації від сервера
POST	Надсилання інформації на сервер
PUT	Додавання інформації до бази даних сервера
DELETE	Видалення інформації з сервера
HEAD	Отримання суто метаданих з сервера
OPTIONS	Пересилання запитів, які обслугує сервер

Продовження таблиці 1.2

Тип запиту	Ціль існування
CONNECT	Прохання до сервера підключити клієнта до іншого сервера (використовується в HTTP over TLS)
TRACE	Отримання в тілі відповіді відправленого запиту
PATCH	Часткова модифікація ресурсу на сервері

На практиці частіше за все використовуються лише GET та POST.

Передача даних за допомогою HTTP виконується через WebAPI – набор адресів сторінок на сайті, які призначені суто для комунікації між програмами (а не для кінцевого користувача), тому пересилають суто дані в зрозумілому для програми форматі (зазвичай JSON).

2 ВИБІР І ОБҐРУНТУВАННЯ ТЕХНІЧНИХ ЗАСОБІВ ДЛЯ СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір мови програмування та технічних засобів

Вибір мови програмування для створення ПЗ – процес, який залежить від багатьох факторів. Нижче будуть оглянуті популярні мови програмування та сфери їх використання.

C/C++ – старі та досить низькорівневі мови, великим плюсом яких є швидкість виконання та висока гнучкість. Сьогодні в основному використовуються для створення ПЗ для вбудованих систем, а також в сферах, де є критичною швидкодійність програми.

Java/C# – дві дуже подібні одна до іншої мови програмування, побудовані на принципі взаємодії з віртуальною машиною. Плюсом такого підходу є значне полегшення роботи з оперативною пам’яттю, але через це теж сильно втрачається швидко діяльність.

JavaScript – мова програмування, створена у першу чергу як скриптова, але завдяки розв’язанням таким як NodeJS може бути використана в розробці окремого від веб-сторінки ПЗ. JS вкрай легко опанувати, але через свою інтерпретованість ця мова дуже повільна.

Python – ще одна скриптова мова. На відміну від JavaScript, вона призначена для праці на сервері. Мінусом Python може також бути низька швидкодійність, але це може бути компенсовано бібліотеками, які використовують посилання до машинного коду (такими як numpy).

Для роботи був обраний Python. Такий вибір було зроблено через низку чинників: дуже широкий вибір бібліотек, простота в опануванні, кросплатформність та висока модульність коду (рис. 2.1).

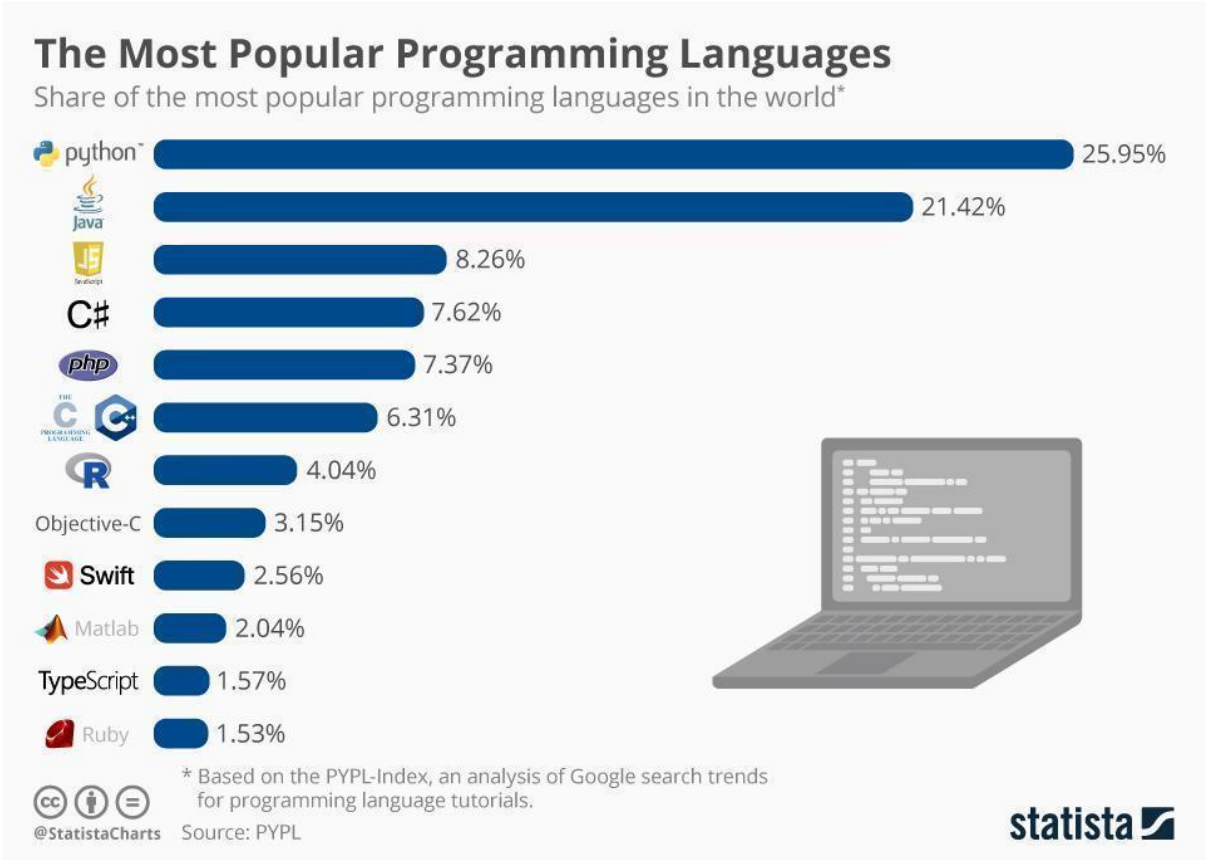

Рисунок 2.1 – Python знаходиться на першому місці серед усіх мов за популярністю [17]

Завдяки своїй простоті та обширному набору бібліотек, Python є найбільш популярною мовою програмування, яку вживають для машинного навчання та аналізу даних. Цьому дуже сприяють бібліотеки-враппери – “обгортки”, що під сподом посилаються на скомпільований машинний код, що значно пришвидшує процес аналізу.

Ще одним аргументом на користь Python є велика кількість backend-фреймворків, які роблять можливим створення найрізноманітніших веб-сторінок.

На сьогодні трьома найбільш популярними веб-фреймворками в Python є Django, Flask та FastAPI.

Django є найпопулярнішим перш за все серед великих фірм, які потребують розбудовані веб-сторінки. Завдяки вбудованій ORM, можливості інтеграції даних безпосередньо до веб-сторінки та системі адміністрації, Django дозволяє швидко будувати складні та великі веб-системи [18].

Flask – це відносно невеликий веб-фреймворк (деколи навіть класифікується як мікрофреймворк). Його дуже легко опанувати: наприклад, для хостингу однієї веб-сторінки достатньо трьох-чотирьох строчок коду [19].

FastAPI – порівняльно новий фреймворк. Він був створений з метою створення простих та видатних API. FastAPI збудований на базі фреймворка Starlette, який дозволяє створювати легкі асинхронні веб застосунки [20]. В табл. 2.1 наведено порівняння характеристик веб-фреймворків.

Таблиця 2.1 – Порівняння характеристик веб-фреймворків

Назва	ORM	Підтримка MVC	Ajax	Тестування	Міграції БД
Django	Так	Так	Так	Так	Так
Flask	ORM-агностичний (можна використати будь-який ORM)	Ні	Так	З бібліотекою pytest	Залежить від ORM
FastAPI	ORM-агностичний (можна використати будь-який ORM)	Ні	Так	З бібліотекою unittest	Залежить від ORM

Для створення роботи обрано FastAPI. Цей фреймворк призначений спеціально для API, завдяки чому містить значно менше зайвого функціоналу, що дозволяє значно пришвидшити роботу програми.

До того, бібліотеки, які входять в склад FastAPI, дозволяють значно полегшити працю з даними.

Pydanic – бібліотека, спрямована на валідацію та серіалізацію даних. Вона дає можливість, отримуючи інформацію в форматі JSON, серіалізувати її, отримуючи готовий об’єкт для обробки. До того ж, Pydantic може легко співпрацювати з інтегрованими середовищами обробки (IDE), що робить розробку ПЗ значно легшою [21].

Pydantic також надає підтримку валідації вхідних даних. За допомогою вбудованої анотації типів мови Python можна позначити, які саме типи можуть приймати поля моделі даних. Таким чином можна запобігти передачі невірної інформації та можливим помилкам сервера, викриваючи та повертаючи помилки в форматі даних.

Більша частина логіки валідації Pydantic не була імплементована в Python – для більш часоємних операцій використовується бібліотека pydantic-core, імплементована в мові програмування Rust, яка примітна швидкістю роботи та використовує мало оперативної пам'яті завдяки мануальному управлінню алокацією. Завдяки цьому Pydantic є дуже видатною бібліотекою [22].

Дуже важливою також є асинхронність сервера. Асинхронне виконання функцій дозволяє їм працювати в окремих потоках, значно пришвидшуючи виконання програми завдяки паралельному виконанню дій.

З цього теж виникають проблеми асинхронних функцій: через те, що декілька функцій виконується одразу, може виникати ситуація, коли дві функції пробують зчитати або записати дані в ту саму частину операційної пам’яті. Це може призвести до неочікуваної поведінки та називається стан гонитви (data race).

Для запобігання цьому можна використати різні методи: наприклад, мутекси – змінні, які блокують виконання частини коду, якщо інша програма заблокувала виконання, а також атомічні змінні, читання та запис до яких займає не більше одного потоку процесора.

Мова Python, як високорівнева, не потребує використання таких методів завдяки бібліотеці asyncio. Завдяки автоматичній оркестрації завдань, асинхронні функції Python не можуть призвести до стану гонитви.

В контексті веб-аплікацій є важливим запобігання завеликому трафіку користувачів. Якщо, наприклад, забагато людей (наприклад, 10000), спробують підключитися до примітивного сервера, його програмне забезпечення може не витримати та припинити правильне функціонування.

Запобігти цьому допомагають спецілалізовані web-сервери (webservers). В мові Python одним з найпопулярніших веб-серверів є uvicorn, який імплементує інтерфейс ASGI (Asynchronous Server Gateway Interface). Завдяки своїй асинхронічності uvicorn значно пришвидшує роботу сервера та запобігає аваріям через занадто великий трафік [23].

Ще один плюс uvicorn, на відміну від більш старих веб серверів, таких як Apache Web Server та nginx, це простота інсталяції. uvicorn можна запустити на будь-якій платформі, яка підтримує Python, та за допомогою нього можна просто, не вимагаючи довгого налаштування, запустити ПЗ, написане на будь-якому веб-фреймворку Python.

Для проекту було вирішено використовувати реляційну базу даних. Це рішення виникає з того, що в структурі програми використовуються статичні моделі даних, які легко можна перекласти на таблицю реляційної бази даних за допомогою ORM.

ORM, або object-relational mapper – додаток до мови програмування, який значно полегшує поєднання та операції з базою даних. Системи ORM дозволяють на базі моделей (в припадку Python це класи) створити таблиці в базі даних та описати їх відношення. Такий підхід називається Code First – об’єкти бази даних створюються на підставі програмного коду. Інший підхід –

Database first – передбачає створення програмного коду об’єктів на підставі вже існуючої бази даних.

FastAPI є ORM-агностичним: дозволяє використовувати будь-яку ORM для пов’язання з базою (на відміну, наприклад, від Django, який має вбудований ORM). Як ORM для створення ПЗ було обрано SQLalchemy. Це рішення було прийнято на підставі того, що SQLalchemy приорітизує повну прозорість спілкування з базою: кожний шаг в обробці даних може бути побачений користувачем та модифікований за бажанням [24].

SQLalchemy складається з двох частин: core (серцевина) та сам ORM. Серцевина абстрагує більшу частину процесу підключення до бази даних та дає можливість полегшити мапування об’єктів Python на таблиці бази даних. Через це часто SQLalchemy часто використовують навіть без ORM, користуючись функціями серцевини до створення запитань до бази даних.

Для комунікації з базою даних також потрібен драйвер бази даних. Драйвер БД – це програмне забезпечення, яке дозволяє за допомогою спеціалізованої бібліотеки виконувати операції на базі даних за допомогою програмного коду. Існують кілька бібліотек, які запевняють підключення до PostgreSQL з рівня язика Python. Найпопулярнішими серед них є psycopg2, asyncpg та вбудована бібліотека SQLalchemy. Для роботи було обрано asyncpg, бо ця бібліотека була створена з метою використання в багатопотоковому програмному заберпеченні. Через це, asyncpg майже в шість рази швидше, ніж її головний суперник psycopg2 (рис. 2.2) [25].

Також важливим компонентом праці з базами даних є міграції. Міграція – це зміна структури бази даних, яка зберігає дані, які містяться в ній, яка є інкрементальною (можна прослідити коли були зроблені ті чи інші зміни), та, зазвичай, яку можна відкотити. Міграції часто використовуються разом з ORM, бо дозволяють вносити зміни в моделі баз даних в процесі розробки, не втрачаючи дані та структуру бази даних.

Для міграцій схеми бази даних SQLalchemy використовує бібліотеку alembic. Ця бібліотека дозволяє створювати автоматизовані запроси до бази даних, та які потім можна відтворити або відмінити.

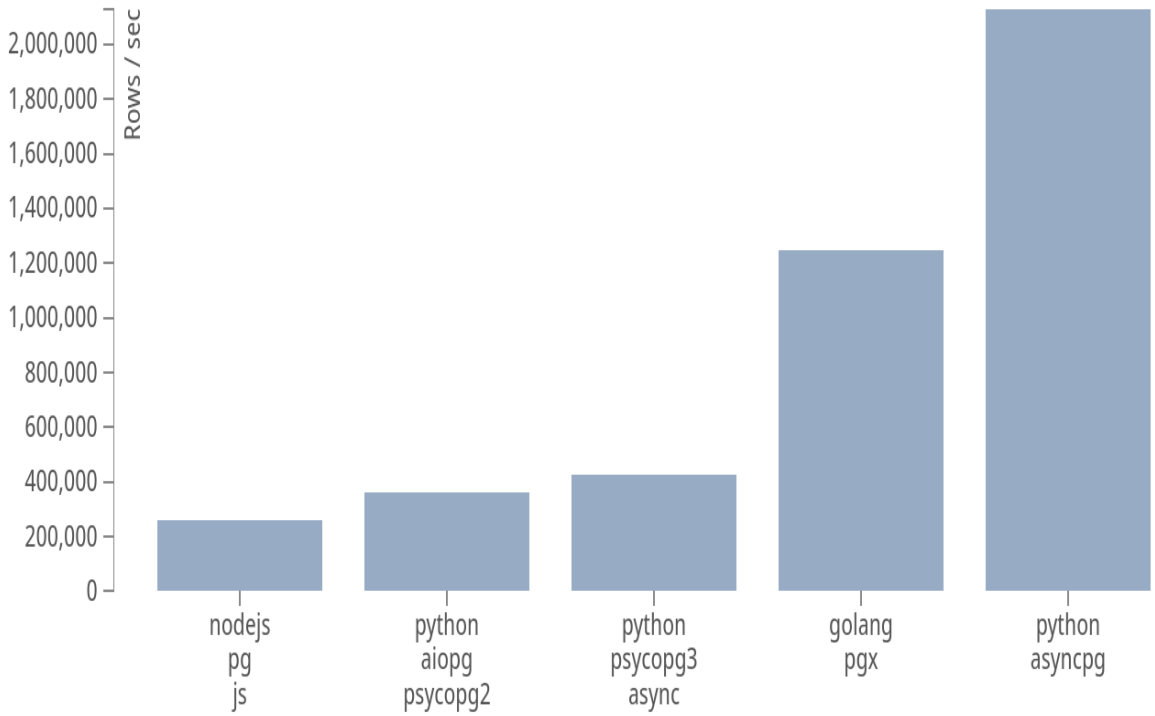

Рисунок 2.2 – Порівняння швидкості різних драйверів баз даних [26]

Дуже важливим в сьогоденні є питання кібербезпеки та авторизації. Веб-сервіси, які недостатньо добре стежать за безпекою та шифруванням своїх даних, ризикують своїми коштами та даними своїх користувачів. Витік даних в достатньо великій компанії може призвести до катастрофічних наслідків для користувачів та репутації фірми. Саме цьому потрібно дбати про правильне та надійне шифрування даних.

В Python для шифрування даних використовується бібліотека cryptography. Ця бібліотека надає доступ до різних алгоритмів шифрування, які забезпечують, що відчитування зашифрованих даних без спеціального ключа стає неможливим. Також вона робить можливим хешування (hashing). Алгоритмі хешування, на відміну від шифру, не дають можливість відчитати інформацію з хешу, але дають можливість перевірити, чи інформація, з якої

отримано хеш, збігається з даною інформацією. Це використовується, наприклад, для зберігання паролів в базах даних: пароль неможливо отримати зломом бази даних, бо в ній зберігається лише хеш, необхідний для перевірки автентичності пароля.

Але як бути, якщо користувач хоче доказати серверу, що він – той, за кого себе видає, без потреби вислання пароля? В такому разі використовуються токени – спеціальні тимчасові коди, які містять в собі інформацію користувача та зашифровані таким чином, що відчитати інформацію може кожна особа за допомогою публічного ключа, але записати може тільки особа, яка посідає приватний ключ. Найбільш поширений тип токенів – JWT, JSON Web Token.

В Python для створення та зчитування JWT служить бібліотека pyjwt. Ця бібліотека дозволяє генерувати токени JWT, зберігаючи в них інформацію користувача та дату, до якої діє токен (це робиться з метою безпеки, щоб зловмисники, які отримають токен, не змогли їм скористатися, наприклад, через добу). Також pyjwt підтримує можливість вибору алгоритму шифрування: так, для кодування використовується алгоритм RS512, який дає можливість записувати JWT за допомогою сертифікатів, які генеруються за допомогою сторонніх програм та є більш надійними, ніж стандартні ключі шифрування.

Для аналізу даних обрано модель GPT-4o-mini. Ця модель є на даний момент одною з найбільш популярний моделей на ринку. Завдяки низькій вартості та досить невеликому обсягу ця модель дозволяє дуже значно прискорити та здешевити процес аналізу даних. Так, ціна запитання до API в GPT-4o становить всього 0.0105 центів, що є невисокою ціною в порівнянні з іншими моделями [27].

Для роботи з LLM було використано бібліотеку openai. Ця бібліотека, створена організацією OpenAI, була створена для полегшення доступу до API продуктів організації. Попри це, openai може також бути використана для підключення до інших LLM, які використовують формат запитань,

підтримуваний openai. openai також підтримує інші, не-текстові формати: за допомогою класу Image можна загружати та отримувати зображення, що можна використати для обробки графічної інформації, а за допомогою класу AudioModel – загружати та аналізувати аудіо-інформацію [28]. В табл. 2.2 наведено порівняння ціни за запит до API відомих моделей.

Таблиця 2.2 – Порівняння ціни за запит до API відомих моделей

Назва моделі	Автор	Ціна за запитання до API
gpt-4o	OpenAI	$0,016
gpt-4o-mini	OpenAI	$0,0105
claude-3-sonnet	Anthropic	$0,0156
gemini-1.5-pro	Google	$0,0112
mistral-large	Mistral	$0,0256

Важливим аспектом програмного забезпечення також є інтерфейс користувача. Попри те, що API може легко бути прочитаний іншою програмою, в випадку кінцевого користувача дані, які пересилаються з API, повинні бути попередньо якимось чином оброблені. В веб-розробці за дуже рідкими виключеннями стосується графічний інтерфейс.

Для створення графічного інтерфейсу користувача можна використати різноманітні інструменти. Наприклад, в останні часи стали популярними веб-застосунки – дуже великі веб-сторінки, які використовують API для того, щоб динамічно змінювати контент веб-сторінки. Такий підхід дозволяє значно заощадити час та об’єм пересилання інформації.

Для створення веб-аплікацій використовуються спеціальні веб-фреймворки, такі як ReactJS та Angular. Але створення таких аплікацій може бути часозатратним та потребувати від користувача значно видатнішого

комп’ютера. Саме тому FastAPI пропонує інше розв’язання – генерування документації OpenAPI та SwaggerUI [29].

Swagger UI – це програмне забезпечення, яке дозволяє користувачу отримати доступ до усіх кінцівок API, які пропонує сервер. Таким чином, користувач може легко протестувати працю тої чи іншої кінцівки перед запуском ПЗ до продукції. Завдяки досить простому інтерфейсу Swagger UI можна використати навіть як інтерфейс користувача.

З оглядом на той факт, що продукт призначений в першу чергу на технічного користувача, як графічний інтерфейс ПЗ було вирішено використати саме Swagger UI. Це дозволить як використовувати функціонал аналізу даних напряму, так і досить легко збудувати, наприклад, сторонній мобільний додаток, користаючись з документації API.

2.2 Схема роботи програмного забезпечення.

ПЗ складається з трьох основних частин: модулю авторизації, модулю користувача та модулю аналізу даних. На початку користувач повинен завторизуватися, використовуючи логін та пароль. Після цього він отримає токен відновлення (refresh token). Це JWT-токен, який має відносно довгий строк життя (зазвичай коло місяця). Цей токен використовується для того, щоб створювати токени доступу (access token).

За допомогою токену доступу користувач може отримати доступ до інших частин системи. Модуль користувача дозволяє йому змінювати дані про себе (аватар, ім’я, пароль тощо). В випадку, якщо користувач забув пароль, його можна змінити за допомогою токену зміни пароля, який надсилається на його адрес e-mail.

Модуль аналізу даних надає користувачу доступ до функціоналу управління даними (рис. 2.3). В ньому містяться чотири ендпоінти:

– аналіз вхідної інформації: цей ендпоінт дозволяє проаналізувати резюме робітника та зробити на цій підставі висновок, яка позиція може йому підійти;

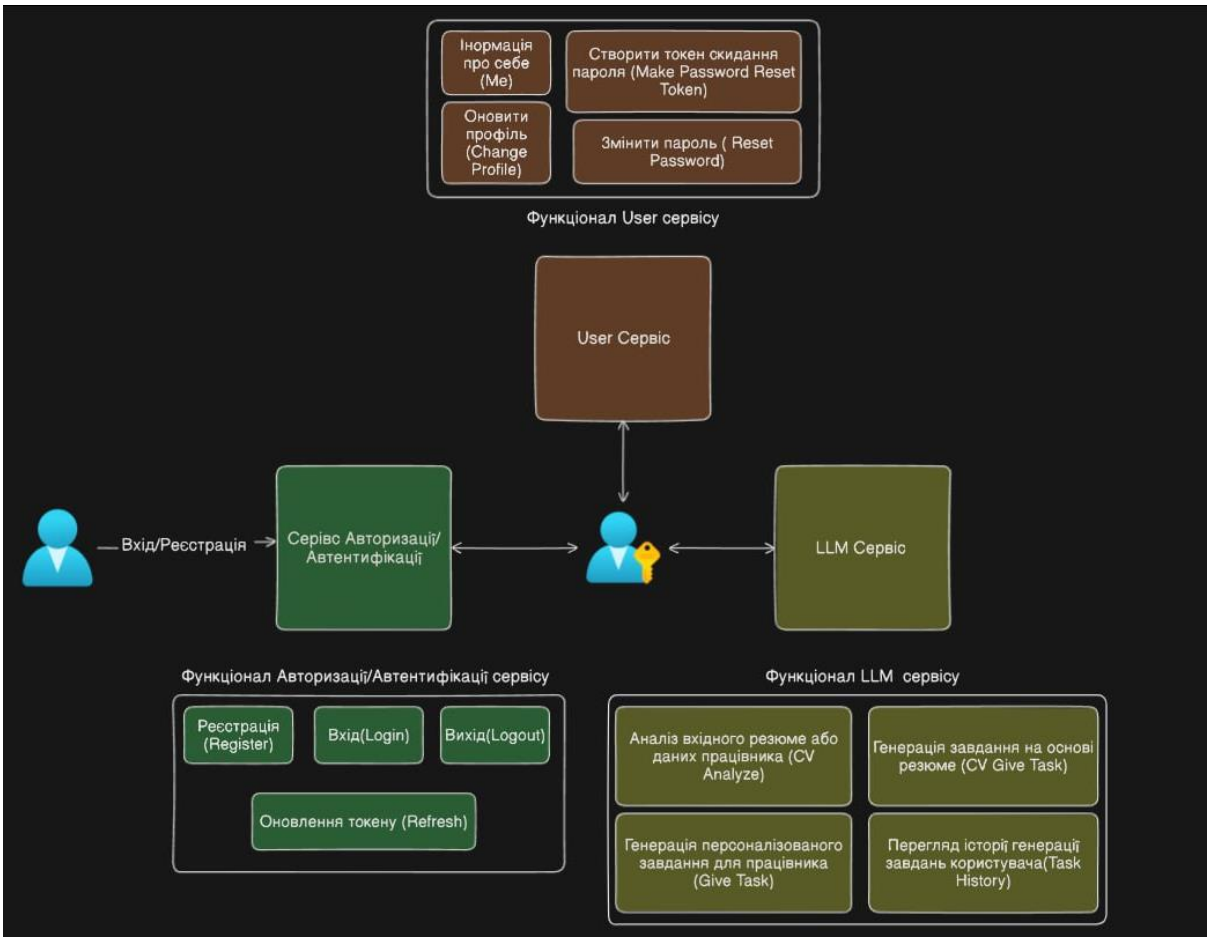

Рисунок 2.3 – Структура програми

– генерація завдань: цей ендпоінт приймає опис працівника (наприклад, отриманий з попереднього ендпоінту) та може вказати, яку саме з доступних задач на виробництві можна йому призначити;

– генерація завдань на підставі CV: цей ендпоінт аналогічний попередньому, але дає завдання працівнику на підставі CV. Це прибирає необхідність додаткового аналізу працівника та дає змогу призначити йому завдання одразу після прийняття на роботу;

– перегляд історії генерації завдань: дає користувачу змогу переглянути завдання, визначені попередньо для різних працівників.

2.3 Прості запити до системи та аналіз об'єктів у файлі

Для авторизації запитів до системи використовується схема refresh+access token. Вона складається з двох токенів: токену відновлення та токену доступу. Токен відновлення має відносно довгий час існування (зазвичай місяць). По цьому часі система змушує користувача знов ввести свої дані для логування.

Токен доступу, в свою чергу, генерується суто по стороні клієнта. Його термін дії становить не більше кількох годин. Після генерації токену доступу він відправляється до сервера, який декодує його та порівнює аутентичність. Це дозволяє завадити тому, що в разі перехоплення передачі сторонні особи можуть за допомогою токена отримати повний доступ до захищених ендпоінтів.

Ендпоінти:

– /auth/login (рис. 2.4): логін користувача

Аутентифікує користувача та генерує новий токен доступу та оновлення. Зберігає токен доступу в сесії.

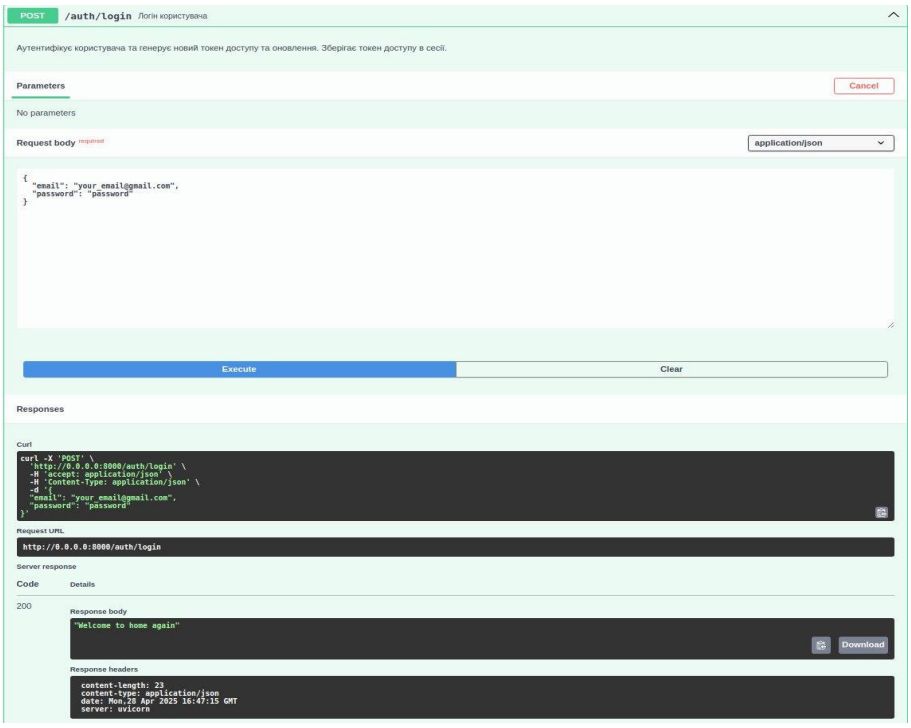

Рисунок 2.4 – Логін користувача /auth/login

– /auth/logout вихід користувача з системи

Виводить користувача з системи, видаляючи його токен оновлення та очищаючи токен доступу до сеансу.

– /auth/refresh

Оновлює токен доступу, використовуючи дійсний токен оновлення. Оновлює сеанс новим токеном.

– /api/ai/cv/analyze (рис 2.5)

Приймає файл (резюме, технічний звіт або форму введення) працівника приладобудівного виробництва та аналізує його за допомогою AI-алгоритму. На основі отриманих даних формує рекомендації щодо компетенцій, навичок та готовності до виконання завдань (рис. 2.5).

Можливі застосування:

– аналіз резюме нового працівника;

– автоматичне визначення рівня технічної підготовки;

– рекомендації щодо призначення завдань.

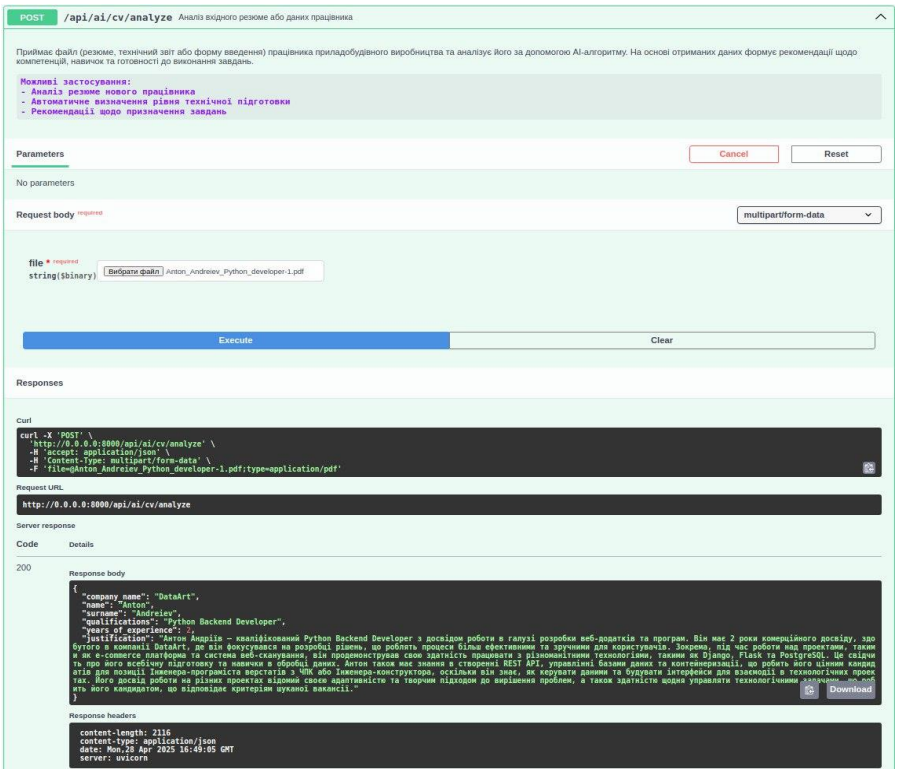

Рисунок 2.5 – Токен доступу /api/ai/cv/analyze

– /api/ai/cv/give_task (рис. 2.6)

Приймає файл з резюме (формати .pdf), аналізує його за допомогою AI-моделі і на основі змісту генерує персоналізоване завдання.

Сценарії використання:

– автоматичне призначення завдань на основі компетенцій з резюме;

– генерація індивідуальних навичок для задачі;

– підтримувані формати: .pdf.

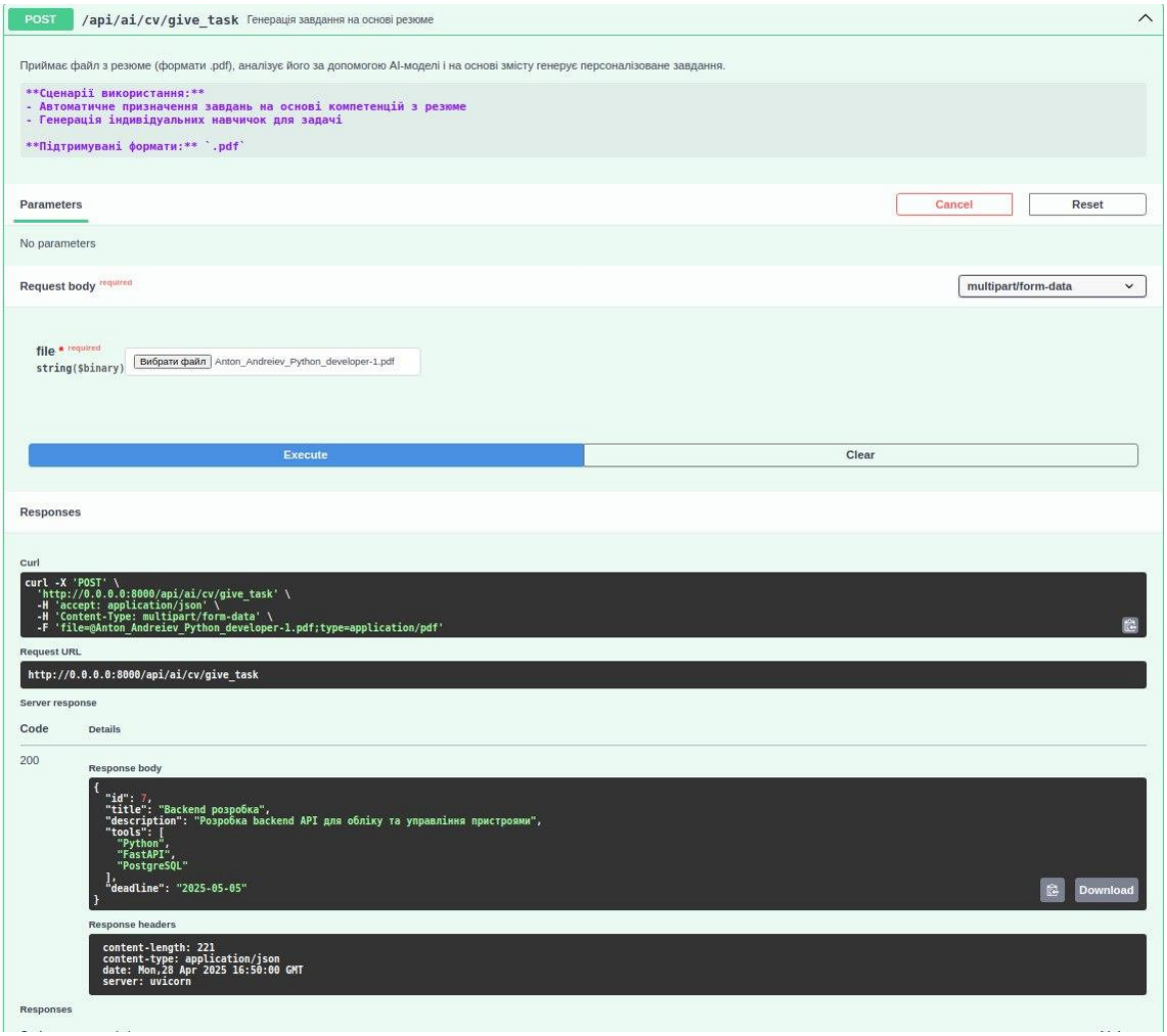

Рисунок 2.6 – /api/ai/cv/give_task

– /api/ai/give_task(рис. 2.7)

Приймає структуровану інформацію про працівника (у вигляді JSON-рядка або опису), аналізує її за допомогою AI та формує персоналізоване технічне або організаційне завдання.

Сценарії використання:

– автоматична видача завдання після аналізу компетенцій;

– генерація індивідуальних навичок для задачі.

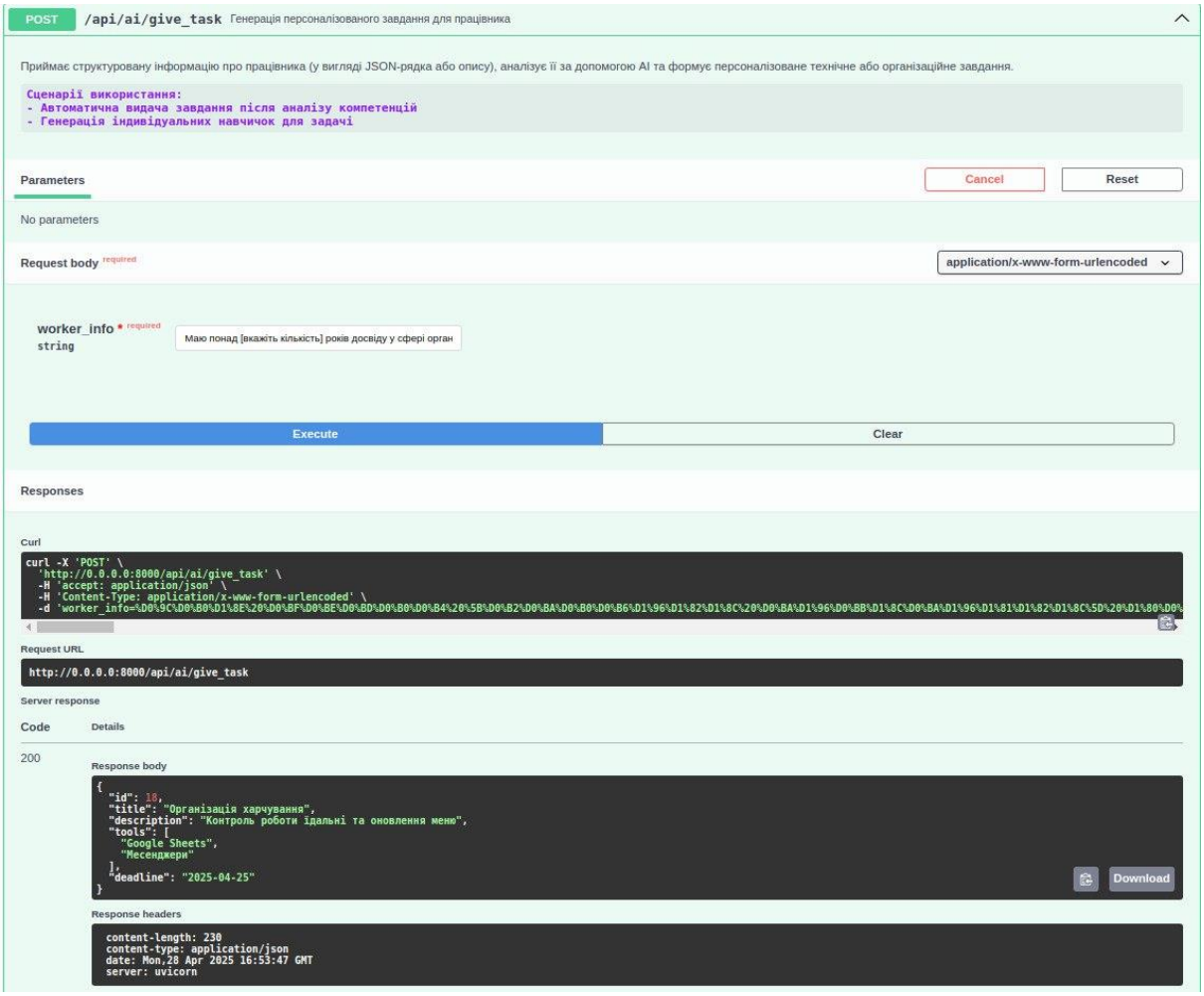

Рисунок 2.7 – /api/ai/give_task

– /api/ai/task_history

Повертає всі запити, що були зроблені користувачем до AI-моделі для генерації персоналізованих завдань, разом з вхідними даними та результатами.

Корисно для:

– внутрішнього аудиту;

– перегляду історії завдань для звітності;

– відтворення ходу прийняття рішень.

2.4 Розрахунки

Оцінимо швидкість роботи серверу. Для цього скористаємося такою формулою:

$$T_{загальний}=T_{клієнт}+T_{мережа}+T_{сервер}, \quad (2.1)$$

де $T_{загальний}$ – загальний час обробки одного запиту від моменту його відправлення клієнтом до отримання відповіді;

$T_{клієнт}$ – час, витрачений на клієнтській стороні;

$T_{мережа}$ – час, витрачений на передачу запиту та отримання відповіді через інтернет;

$T_{сервер}$ – час, витрачений на обробку запиту сервером.

Розглянемо детальніше кожну складову:

час на клієнтській стороні ($T_{клієнт}$):

$$T_{клієнт}=T_{підготовка\_запиту}+T_{очікування\_відповіді}, \quad (2.2)$$

де $T_{підготовка\_запиту}$ – час, необхідний клієнтському застосунку для створення та підготовки запиту до відправлення (наприклад, формування HTTP-запиту, серіалізація даних). Цей час залежить від складності запиту, продуктивності клієнтського пристрою та ефективності клієнтського коду;

$T_{очікування\_відповіді}$ – частина загального часу, протягом якої клієнтський застосунок очікує на отримання відповіді від сервера. Ця складова вже

включена в $T_{загальний}$, тому при окремому розрахунку швидкості обробки запитів (кількість запитів за одиницю часу) її враховувати не потрібно.

Час передачі через інтернет ($T_{мережа}$):

$$T_{мережа}=T_{передача\_запиту}+T_{затримка\_мережі}+T_{передача\_відповіді}, \quad (2.3)$$

де $T_{передача\_запиту}$ – час, необхідний для передачі запиту від клієнта до сервера через інтернет. Залежить від розміру запиту та швидкості інтернет-з'єднання клієнта;

$T_{затримка\_мережі}$ (latency) – час, необхідний пакету даних, щоб пройти через мережу (маршрутизатори, комутатори тощо). Залежить від відстані між клієнтом і сервером, кількості проміжних вузлів та завантаженості мережі. Цей час може включати ping;

$T_{передача\_відповіді}$ – час, необхідний для передачі відповіді від сервера до клієнта через інтернет. Залежить від розміру відповіді та швидкості інтернет-з'єднання клієнта (а також сервера).

Час обробки на сервері ($T_{сервер}$):

$$T_{сервер}=T_{прийом\_запиту}+T_{обробка\_запиту}+T_{підготовка\_відповіді}, \quad (2.4)$$

де $T_{прийом\_запиту}$ – час, необхідний серверу для отримання та розбору запиту. Залежить від завантаженості сервера та протоколу зв'язку;

$T_{обробка\_запиту}$ – основний час, витрачений сервером на виконання запиту (наприклад, виконання запиту до бази даних, обчислення, звернення до інших сервісів). Цей час критично залежить від складності запиту, продуктивності сервера, оптимізації серверного коду та завантаженості сервера;

$T_{підготовка\_відповіді}$ – час, необхідний серверу для формування та підготовки відповіді до відправлення (наприклад, серіалізація даних, формування HTTP-відповіді).

Тепер проведемо розрахунки:

$$T_{\text{підготовка запиту}} = 3 \text{ мс}, \quad (2.5)$$

$$T_{\text{очікування відповіді}} = 11{,}3 \text{ мс}, \quad (2.6)$$

$$T_{\text{передача запиту}} = 3{,}4 \text{ мс}, \quad (2.7)$$

$$T_{\text{затримка мережі}} = 2{,}5 \text{ мс}, \quad (2.8)$$

$$T_{\text{передача відповіді}} = 1{,}7 \text{ мс}, \quad (2.9)$$

$$T_{\text{прийом запиту}} = 2{,}0 \text{ мс}, \quad (2.10)$$

$$T_{\text{обробка запиту}} = 1{,}7 \text{ мс}, \quad (2.11)$$

$$T_{\text{підготовка відповіді}} = 0{,}042 \text{ мс}, \quad (2.12)$$

$$T_{\text{клієнт}} = T_{\text{підготовка запиту}} + T_{\text{очікування відповіді}} = 14{,}3 \text{ мс}, \quad (2.13)$$

$$T_{\text{мережа}} = T_{\text{передача запиту}} + T_{\text{затримка мережі}} + T_{\text{передача відповіді}} = 7{,}6 \text{ мс}, \quad (2.14)$$

$$T_{\text{сервер}} = T_{\text{прийом запиту}} + T_{\text{обробка запиту}} + T_{\text{підготовка відповіді}} = 3{,}742 \text{ мс}, \quad (2.15)$$

$$T_{\text{загальний}} = T_{\text{клієнт}} + T_{\text{мережа}} + T_{\text{сервер}} = 14{,}3 \text{ мс}. \quad (2.16)$$

Формула для швидкості обробки запитів (Requests Per Second – RPS): швидкість обробки запитів можна визначити як кількість успішно оброблених запитів за одиницю часу. Якщо ми маємо загальний час обробки одного запиту ($T_{\text{загальний}}$ в секундах), то теоретична максимальна швидкість обробки одного запиту (з точки зору одного запиту) буде:

швидкість_одного_запиту = 1/ $T_{загальний}$ запитів/секунду. (2.17)

Однак, на практиці, загальна швидкість обробки запитів системою залежить від багатьох паралельних факторів, особливо від здатності сервера обробляти одночасні запити. Тому більш практична формула для вимірювання загальної швидкості обробки запитів протягом певного періоду часу (t) буде:

$$\text{RPS} = \text{Кількість\_успішно\_оброблених\_запитів} / t \text{ (в секундах)}, \quad (2.18)$$

$$\text{RPS} = 88 \text{ запитів}/1c. \quad (2.19)$$

3 РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Блок-схема роботи програми

Спочатку система авторизує користувача, використовуючи токен сесії. Якщо користувач не посідає такого токену, система повідомлює його про необхідність нової авторизації.

У випадку, якщо токен був переданий, але його термін дії сплив, здійснюється спроба створення нового токену сесії за допомогою токена відновлення.

Якщо токен відновлення дійсний, користувачу передається новий створений токен сесії для подальшого використання, в іншому випадку необхідно знову авторизуватися.

Після отримання користувачем дійсного токену сесії система перевіряє, який саме запит було здійснено.

У випадку ендпоінтів /cv/analyze, /cv/give_task та /give_task, виконується додаткова перевірка на правильність введених даних. Якщо формат даних неправильний, користувач отримує помилку 400 (Bad Request).

У випадку ендпоінта /task_history такої перевірки робити не треба. У випадку, якщо запитання користувача не збігається з жадним з ендпоінтів, користувач отримує помилку 404 (Not Found).

Якщо запит правильний, користувач отримує http-відповідь з кодом 200, яка містить відповідні запиту дані.

Результат алгоритму зображено на рис. 3.1 та рис. 3.2.

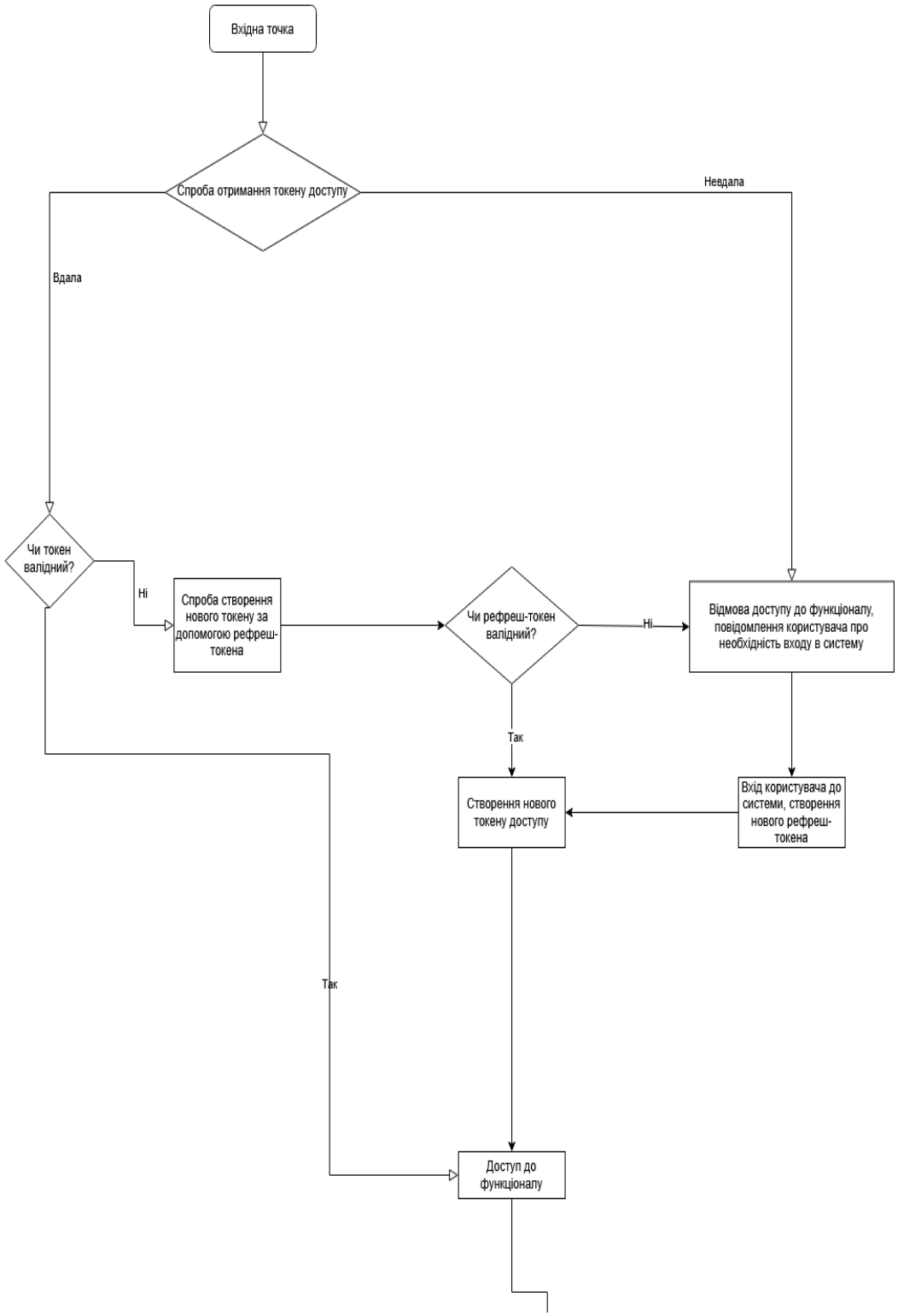

Рисунок 3.1 – Блок-схема роботи програми

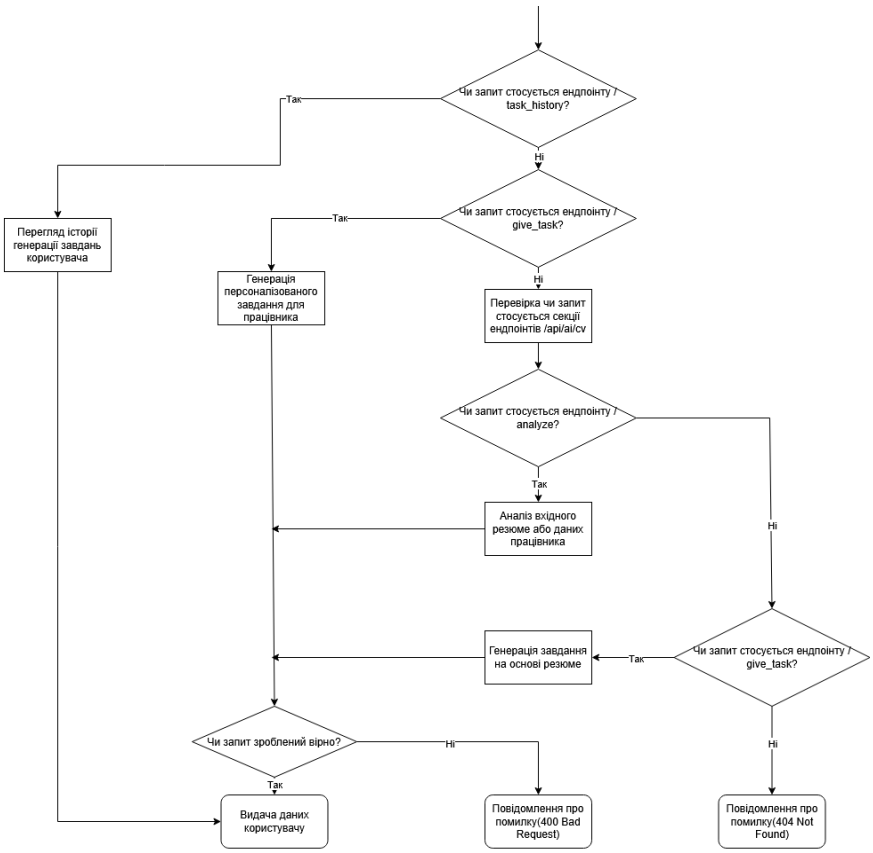

Рисунок 3.1, аркуш 2

На цій блок-схемі можна побачити, що програма не може допустити користувача до функціоналу програми перед тим, як він авторизується в системі.

3.2 Розроблення додатку

Спочатку створимо розв’язання FastAPI. Вхідною точкою до додатку буде файл main.py. Програма починає працювати в точці, в якій викликано функцію get_application.

Почнемо з підключення бібліотек:

```
import logging
import sys
```

```
from contextlib import asynccontextmanager
from fastapi import FastAPI, Request, status
from starlette.middleware.sessions import SessionMiddleware
from starlette.responses import JSONResponse

from app.api.routers.users import router as user_router
from app.api.routers.auth import router as auth_router
from app.api.routers.ai import router as ai_router
from app.config import settings, config
from app.database import sessionmanager
```

Секцію підключення можна поділити на дві частини: підключення сторонніх бібліотек та підключення внутрішніх файлів. Треба додати, що внутрішні файли теж можуть підключати свої бібліотеки.

Тепер підключимо бібліотеку logger, яка є стандартною бібліотекою для запису даних статусу додатку. Це може допомогти в випадку, коли додаток видасть несподівану поведінку: дії, які виконує додаток, записуються до файлової системи.

```
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG if settings.log_level == "DEBUG" else logging.INFO)

root_logger = logging.getLogger("root_logger")
```

Створимо функцію get_application, яка буде вхідною точкою додатку та додасть відповідню конфігурацію:

```
def get_application(settings_db: str = "", init_db: bool = True) -> FastAPI:
"""
Створення головного додатку FastAPI
:param settings_db: класс із наштуваннями для БД (Postgres)
```

:param init_db: bool значення яке використовується як парамет при створенні додатку із використанням БД або без

:return:

"""

Функція повертає об’єкт типу FastAPI, який репрезентує веб-сервер. Цей об’єкт потім може бути налаштований, розширюючи функціонал додатку.

```
if init_db:
    sessionmanager.init(settings_db)

    @asynccontextmanager
    async def lifespan(app: FastAPI):
    yield
    if sessionmanager._engine is not None:
        await sessionmanager.close()
```

Змінна init_db визначає необхідність ініціалізації бази даних. Якщо вона вставновлена на false, базу даних не буде створено та використано в додатку.

Тепер створимо об’єкт FastAPI, таким чином запускаючи додаток:

```
app = FastAPI(
    lifespan=lifespan,
    title="Система авторизації та AI Hepler для приладобудівного виробництва",
    description="""
    🔐 **Опис проєкту:**

    Даний застосунок є частиною дипломного проєкту та реалізує **надійне, продумане і структуроване рішення для автентифікації та авторизації користувачів** у корпоративному середовищі. Система забезпечує високий рівень безпеки та контроль доступу до функцій і даних на основі ролей.
```

🧠 **Призначення:**

Розроблено як бекенд-модуль для **аналізу вхідної інформації працівника приладобудівного виробництва** з метою:

– обробки та інтерпретації інформації від користувачів системи;
– генерації завдань на основі поведінки, ролі та статусу працівника;
– централізованого керування доступом до функціоналу системи.

⚙️ **Основний функціонал API:**

– ✅ реєстрація користувача з валідацією даних;
– 🔐 вхід з перевіркою облікових даних;
– 🚪 вихід із системи з анулюванням токенів;
– ♻️ оновлення access токену через refresh токен;
– 📊 логування дій користувача для подальшого аудиту.

🛡️ **Безпека:**

– json Web Tokens (JWT);
– сесійне керування;
– механізми перевірки типу токена;
– логіка ротації refresh-токенів.

📦 **Формат обміну даними:** Всі запити й відповіді структуровано у форматі JSON для REST API-інтерфейсу.

🖼️ **Призначено для використання у внутрішніх системах підприємств у сфері приладобудування**, де потрібне чітке управління

користувачами, правами доступу та видачею технологічних або організаційних завдань.

```
        """,
        version="1.0.0",
        contact={
        "name": "Anton Andreiev",
        "email": "anton@example.com"
        },
        license_info={
        "name": "MIT License",
        "url": "https://opensource.org/licenses/MIT",
        },
        docs_url="/api/docs",
        redoc_url="/api/redoc"
)
```

Конструктор об’єкту FastAPI приймає низку аргументів, але в цьому випадку використовуються лише lifespan, title, description, version, contact, license_info, docs_url та redoc_url.

Розберемо значення параметрів:

– lifespan є менеджером контексту додатку. Це означає, що lifespan описує, які дії мають бути виконані перед початком та після закінчення праці. В цьому випадку програма має завершити роботу менеджера сесії бази даних після закінчення;

– title, description – це назва та опис проекту, які з’являться на його головній сторінці;

– version – версія проекту;

– contact – контактні дані автора веб-сторінки;

– docs_url – url, за яким можна буде знайти документацію сторінки. Саме за допомогою нього користувач зможе отримати доступ до функціоналу застосунку.

– redoc_url – це url, за яким можна знайти альтернативну версію документації. Ця версія використовує ReDoc, на відміну від SwaggerUI, та пропонує дещо інший функціонал.

Тепер додамо middleware. Middleware, або проміжне ПЗ – це шари програмного забезпечення, які можна вставити поміж різними компонентами ПЗ, не змінюючи основного функціоналу цих компонентів. В веб-розробці middleware використовується, щоб легко додати якийсь функціонал до ендпоінту. Наприклад, найчастіше застосування middleware – авторизація. Middleware дозволяє не імплементувати авторизацію “з нуля”, а просто створити проміжне ПЗ, яке буде перехоплювати запроси на “захищені” ендпоінти, та повертати помилку (найчастіше 403 – unauthorized), якщо користувач не авторизований.

app.add_middleware(SessionMiddleware, secret_key="some-random-string")

Також додамо middleware, яке буде реагувати на неправильно сформульовані реквести:

```
@app.middleware("http")
async def exception_handling(request: Request, call_next):
    root_logger.info(
    f"Url: {request.url}\n"
    f"Header: {request.headers}\n"
    f"Method: {request.method}\n"
    f"Query params: {request.query_params}\n"
    f"Path params: {request.path_params}\n"
    )
    try:
    return await call_next(request)
    except Exception as exc:
```

```
        root_logger.error(str(exc), exc_info=True)
        return JSONResponse(
                status_code=status.HTTP_400_BAD_REQUEST,
                content={"detail": str(exc)},
        )
```

FastAPI дозволяє додавати ендпоінти дуже швидко та просто. Додати простий ендпоінт можна всього за кілька строк коду:

```
@app.get("/")
async def root():
   return {"message": "Hello World"}
```

Але додавання всіх ендпоінтів в одному файлі стає неможливим, коли кількість ендпоінтів стає завеликою та утримання коду стає все труднішим. Тому використаємо механізм роутерів – підпрограм, які займаються управлінням окремими частинами веб-застосунку.

```
app.include_router(user_router)
app.include_router(auth_router)
app.include_router(ai_router)
```

Тепер завершимо роботу функції, повертаючи об’єкт FastAPI.

```
return app


app = get_application(settings.database_config.DB_CONFIG[0])
```

Тепер розглянемо роутери, які відповідають за API-функціонал програми: файл user.py відповідає модулю управління користувачами. В ньому імплементовані ендпоінти, які відповідають за отримання та зміну даних користувача.

Підключення бібліотек:

```
from uuid import UUID
from fastapi import APIRouter, Depends, Request, HTTPException
from typing import Annotated

from app.api.dependencies.auth import validate_password_reset
from app.api.dependencies.user import CurrentUserDep, CurrentAdminDep
from app.api.dependencies.core import DBSessionDep
from app.crud.log import create_log
from app.crud.user import (
update_user_profile,
create_password_token,
create_new_password
)
from app.schemas.user import (
User,
AuthorizedUser,
UpdateProfile,
ResetPasswordArgs
)
```

Важливо сказати, про то, що більшість функціоналу, не пов’язана з суто обробкою http-запитів, була винесена в окремі файли того ж проекту. Це було зроблено, по-перше, для полегшення роботи над програмою, та, по-друге, забезпечує краще розділення відповідальностей.

Створення об’єкту роутера:

```
router = APIRouter(
prefix="/api/users",
tags=["Users Control System"],
```

```
responses={404: {"description": "Ресурс не знайдено"}}
)
```

До роутера можна додати ендпоінт за допомогою декоратора @router. + тип http-запиту. Наприклад:

```
@router.get(
"/me",
summary="Отримати дані поточного користувача",
description="Повертає детальну інформацію про поточного авторизованого користувача, включаючи ID, email, ім'я, роль та дату створення облікового запису.",
response_model=AuthorizedUser,
response_description="Інформація про поточного користувача",
responses={
    200: {"description": "Успішне отримання даних користувача"},
    401: {"description": "Користувач не авторизований"},
    500: {"description": "Внутрішня помилка сервера"}
}
)
async def user_details(current_user: CurrentUserDep, db_session: DBSessionDep):
await create_log(db_session, "me", current_user)
return current_user
```

Такий код додасть ендпоінт, який приймає get-запит за адресою /api/users/me. Також в декораторі можна налаштувати опис ендпоінту та інформацію щодо даних, які він повертає.

current_user: об'єкт користувача з Depends.

db_session: сесія бази даних.

Повертає: pydantic-схему `AuthorizedUser`.

```
@router.patch(
"/change/profile",
summary="Змінити профіль користувача",
description="Оновлює профіль поточного користувача. Доступно редагування імені, прізвища, тощо.",
response_model=AuthorizedUser,
response_description="Оновлений профіль користувача",
responses={
    200: {"description": "Профіль успішно оновлено"},
    400: {"description": "Помилка валідації або неправильні дані"},
    401: {"description": "Користувач не авторизований"},
    500: {"description": "Внутрішня помилка сервера"}
}
)
async def change_profile(
    profile_update: UpdateProfile,
    current_user: CurrentUserDep,
    db_session: DBSessionDep
):
try:
    updated_user = await update_user_profile(db_session, current_user, profile_update)
except ValueError as e:
    raise HTTPException(status_code=400, detail=str(e))

await create_log(db_session, "change_profile", current_user)
return updated_user
```

Змінює особисті дані поточного користувача.

profile_update: нові дані профілю (ім'я, прізвище тощо)

current_user: авторизований користувач

Повертає: оновлений об'єкт користувача

Завдяки використанню метода patch немає потреби пересилання цілого профілю користувача, що значно зменшує використання даних.

Тепер створимо ендпоінти, пов’язані з токеном та паролем.

```
@router.post(
"/make/password_reset_token",
summary="Створити токен скидання пароля",
description="Генерує токен для скидання пароля поточного користувача. Токен буде прив'язаний до його облікового запису.",
response_description="Повідомлення про успішне створення токена",
responses={
    200: {"description": "Токен успішно створено"},
    401: {"description": "Користувач не авторизований"},
    500: {"description": "Помилка при створенні токена"}
}
)
async def make_password_reset_token(
    current_user: CurrentUserDep,
    db_session: DBSessionDep
):
await create_password_token(db_session, current_user)
await create_log(db_session, "make_password_reset_token", current_user)
return "Create success new password token"
```

Створює токен для скидання пароля користувача.

current_user: поточний користувач.
Повертає: текстове повідомлення про успішність операції.

```
async def reset_password(
        reset_password_args: ResetPasswordArgs,
        db_session: DBSessionDep,
        current_user: User = Depends(validate_password_reset),
):
await create_new_password(db_session, current_user, reset_password_args)
await create_log(db_session, "reset_password", current_user)
return {"Success": True}
```

Скидає пароль користувача з використанням токена.

reset_password_args**: новий пароль і токен
current_user**: користувач, який має дійсний токен
Повертає: JSON з підтвердженням `{"Success": True}`

```
@router.get(
"/admin",
summary="Отримати дані адміністратора",
description="Повертає дані поточного адміністратора (якщо користувач має відповідні права доступу).",
response_model=AuthorizedUser,
response_description="Інформація про адміністратора",
responses={
        200: {"description": "Успішне отримання даних адміністратора"},
        403: {"description": "Доступ заборонено"},
```

```
        401: {"description": "Користувач не авторизований"}
    }
    )
    async def user_details(current_admin: CurrentAdminDep):
    return current_admin
```

Повертає інформацію про адміністратора.

current_admin: поточний адміністратор
Повертає: pydantic-схему `AuthorizedUser`

Тепер розглянемо інші роутери. Почнемо з роутера auth, який містить в собі функціонал, необхідний для авторизації та аутентифікації користувача. Процес аутентифікації в програмі виглядає наступним чином: при спробі логування сервер робить запит до сесії користувача та пробує отримати токен доступу. Якщо користувач не має такого токену – система попросить його ввести дані (логін та пароль). Після цього система створить новий токен відновлення (рефреш-токен) та на його основі – новий токен доступу, який буде вислано користувачу. В випадку, якщо користувач подав вірний токен доступу та цей токен збігається з токеном, який зберігається в базі даних, робиться перевірка, чи він не вичерпався (зазвичай термін життя такого токену – до одної доби). Якщо токен вичерпався – непомітно для кінцевого користувача система створює новий токен на підставі рефреш-токена та відновлює дані в базі даних. Після цього користувач може отримати доступ до “захищених” ендпоінтів.

В випадку ж, якщо сплине строк дії токена відновлення, користувача просять ще раз подати логін та пароль. Зазвичай це робиться щомісяця.

```
import logging
from datetime import timedelta
```

```
import jwt
from fastapi import APIRouter, Depends, Request, HTTPException, status
from fastapi.responses import JSONResponse

from app.crud.log import create_log
from app.schemas.auth import TokenData
from app.api.dependencies.core import DBSessionDep
from app.api.dependencies.user import CurrentUserDep, CurrentRefreshDataUser
from app.api.dependencies.auth import validate_signup, validate_login, validate_is_authenticated
from app.schemas.auth import Signup, Token, TokenInfo, LoginValidationResult
from app.crud.user import create_user, get_user_by_email
from app.crud.auth import create_refresh_auth_token, update_refresh_auth_token, delete_refresh_auth_token, \
handle_refresh_token, get_refresh_auth_token, get_refresh_auth_token_without_user
from app.models.user import User
from app.constants import REFRESH_TOKEN_TYPE
from app.services.auth import create_access_token, create_refresh_token, get_token_payload, validate_token_type

router = APIRouter(prefix="/auth", tags=["Auth System"])


@router.post(
"/registration",
summary="Реєстрація користувачів",
```

```
    description="Створює нового користувача з заданими обліковими даними і повертає ідентифікатор користувача, ім'я користувача та час створення.",
    response_description="Результат реєстрації з основними даними користувача",
    responses={
        200: {"description": "Користувача успішно зареєстровано."},
        400: {"description": "Неправильні дані для реєстрації."},
        500: {"description": "Внутрішня помилка сервера."}
    }
    )
    async def signup(
        request: Request,
        db_session: DBSessionDep,
        signup: Signup = Depends(validate_signup),
    ):
    user = await create_user(db_session, signup)

    refresh_token = await create_refresh_auth_token(db_session, user)
    access_token = create_access_token(user, refresh_token)

    request.session["access_token"] = access_token

    await create_log(db_session, "signup", user)

    return {
        "user_id": user.id,
        "user_name": user.username,
        "time_creation": user.created
    }
```

Обробляє реєстрацію користувачів.

signup: підтверджені дані для реєстрації (email, пароль, ім'я користувача тощо.)

Повертає: ідентифікатор користувача, ім'я користувача та мітка часу створення.

```
@router.post(
"/login",
summary="Логін користувача",
description="Аутентифікує користувача та генерує новий токен доступу та оновлення. Зберігає токен доступу в сесії.",
response_description="Привітальне повідомлення після успішного входу",
responses={
        200: {"description": "Успішний вхід."},
        401: {"description": "Недійсні облікові дані."},
        500: {"description": "Внутрішня помилка сервера."}
}
)
async def login(
        request: Request,
        db_session: DBSessionDep,
        login_result: LoginValidationResult = Depends(validate_login)
):
user = login_result.user
refresh_token_id = login_result.refresh_token_id
```

refresh_token = await handle_refresh_token(db_session, user, refresh_token_id)

access_token = create_access_token(user, refresh_token)
request.session["access_token"] = access_token

await create_log(db_session, "login", user)

return "Welcome to home again"

Перевіряє автентичність облікових даних користувача.

login_result: містить екземпляр користувача та оновлений ідентифікатор токена після валідації.

Повертає: вітальне повідомлення та встановлює токен доступу до сеансу.

@router.post(
"/logout",
summary="Вихід користувача з системи",
description="Виводить користувача з системи, видаляючи його токен оновлення та очищаючи токен доступу до сеансу.",
response_description="Повідомлення про успішний вихід з системи",
responses={
    200: {"description": "Вихід успішно завершено."},
    401: {"description": "Неавторизований або відсутній токен оновлення."},
    500: {"description": "Внутрішня помилка сервера."}
}

```
)
async def logout(
        request: Request,
        db_session: DBSessionDep,
        refresh_data: CurrentRefreshDataUser
):
user = refresh_data.user
refresh_token_id = refresh_data.refresh_token_id

await delete_refresh_auth_token(db_session, user, refresh_token_id)
del request.session["access_token"]

await create_log(db_session, "refresh", None)
return "Logout successful"
```

Виводить користувача з системи, анулюючи токен оновлення.

refresh_data: включає поточний ідентифікатор користувача та ідентифікатор токена оновлення.

Повертає: підтвердження повідомлення.

```
@router.post(
"/refresh",
summary="Оновити маркер доступу",
description="Оновлює токен доступу, використовуючи дійсний токен оновлення. Оновлює сеанс новим токеном.",
response_description="Новий токен доступу",
responses={
        200: {"description": "Токен доступу оновлено."},
```

```
            401: {"description": "Недійсний або прострочений токен оновлення."},
            500: {"description": "Внутрішня помилка сервера."}
        }
        )
        async def auth_refresh_token(
            request: Request,
            db_session: DBSessionDep,
            refresh_data: CurrentRefreshDataUser
        ):
        try:
            refresh_token = await update_refresh_auth_token(db_session, refresh_data.user, refresh_data.refresh_token_id)
            access_token = create_access_token(refresh_data.user, refresh_token)
        except Exception as e:
            logging.error(f"Refresh token error: {e}")
            raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid refresh token.")

        request.session["access_token"] = access_token

        await create_log(db_session, "refresh", refresh_data.user)

        return access_token
```

Генерує новий токен доступу з дійсного токена оновлення.

refresh_data: містить ідентифікатор користувача та його токену оновлення.

Повертає: новий рядок токену доступу.

Тепер розглянемо роутер, який відповідає за аналіз даних та штучний інтелект. Як і в випадку двох попередніх роутерів, функціонал (бізнес-логіку) винесено в окремі функції, щоб запобігти занадто великому розміру файлів.

```
from fastapi import APIRouter, Depends, Request, HTTPException, UploadFile, File, status, Form

from app.api.dependencies.core import DBSessionDep
from app.api.dependencies.user import CurrentUserDep
from app.crud.request_log import create_request_log, get_request_logs_user
from app.models import TaskRequestLog
from app.services.data_anls import get_worker_statistics, get_worker_data, get_task_by_cv

router = APIRouter(
prefix="/api/ai",
tags=["AI"],
responses={404: {"description": "Не знайдено"}}
)


@router.post(
"/cv/analyze",
summary="Аналіз вхідного резюме або даних працівника",
description="""
Приймає файл (резюме, технічний звіт або форму введення) працівника приладобудівного виробництва
та аналізує його за допомогою AI-алгоритму. На основі отриманих даних формує рекомендації щодо
```

компетенцій, навичок та готовності до виконання завдань.

Можливі застосування:

– аналіз резюме нового працівника;

– автоматичне визначення рівня технічної підготовки;

– рекомендації щодо призначення завдань.

""",

response_description="AI-висновок щодо працівника",

responses={

200: {"description": "Аналіз виконано успішно"},

400: {"description": "Помилка у файлі або форматі"},

500: {"description": "Внутрішня помилка при обробці AI-моделлю"}

}

)

async def analyze_cv(

current_user: CurrentUserDep,

db_session: DBSessionDep,

file: UploadFile = File(...)

):

feedback, content = await get_worker_statistics(file)

await create_request_log(db_session, current_user, content, feedback)

return feedback

Виконує AI-аналіз завантаженого файлу працівника.

file: Файл формату .pdf/.docx/.txt, що містить резюме або опис дій працівника

current_user: авторизований користувач, що ініціює аналіз

Повертає: json з оцінкою та висновками AI

@router.post(
"/cv/give_task",
summary="Генерація завдання на основі резюме",
description="""
Приймає файл з резюме (формати .pdf),
аналізує його за допомогою AI-моделі і на основі змісту генерує персоналізоване завдання.

Сценарії використання:
– автоматичне призначення завдань на основі компетенцій з резюме;
– генерація індивідуальних навчичок для задачі.

Підтримувані формати: `.pdf`
""",
response_description="AI-завдання, згенероване на основі аналізу резюме",
responses={
200: {"description": "Завдання успішно згенеровано"},
400: {"description": "Файл не відповідає вимогам або містить недостатньо інформації"},
415: {"description": "Непідтримуваний формат файлу"},
500: {"description": "Внутрішня помилка при обробці файлу або генерації"}
}
)
async def get_task_by_cv(
current_user: CurrentUserDep,

```
db_session: DBSessionDep,
file: UploadFile = File(...)
):
feedback, content = await get_task_by_cv(file)
await create_request_log(db_session, current_user, content, feedback)
return feedback
```

Генерує AI-завдання на основі завантаженого резюме працівника.

file: завантажений файл з резюме або профілем працівника (.pdf)

current_user: авторизований користувач, що ініціює запит

Повертає: згенероване персоналізоване завдання

```
@router.post(
"/give_task",
summary="Генерація персоналізованого завдання для працівника",
description="""
```

Приймає структуровану інформацію про працівника (у вигляді JSON-рядка або опису),

аналізує її за допомогою AI та формує персоналізоване технічне або організаційне завдання.

Сценарії використання:

– автоматична видача завдання після аналізу компетенцій;

– генерація індивідуальних навчичок для задачі.

```
""",
response_description="Згенероване завдання",
responses={
    200: {"description": "Завдання успішно сформовано"},
    400: {"description": "Недостатньо інформації або помилка у форматі"},
    500: {"description": "Помилка при генерації завдання"}
```

```
}
)
async def give_task(
current_user: CurrentUserDep,
db_session: DBSessionDep,
worker_info: str = Form()
):
feedback = await get_worker_data(worker_info)

# Логування
await create_request_log(db_session, current_user, worker_info, feedback)
return feedback
```

Генерує завдання на основі вхідних даних про працівника.

worker_info: json-рядок або опис даних працівника
current_user: користувач, який виконує запит
Повертає: результат аналізу у вигляді завдання

```
@router.get(
"/task_history",
summary="Перегляд історії генерації завдань користувача",
description="""
```

Повертає всі запити, що були зроблені користувачем до AI-моделі для генерації персоналізованих завдань,
разом з вхідними даними та результатами.

Корисно для:
– внутрішнього аудиту;

– перегляду історії завдань для звітності;

– відтворення ходу прийняття рішень.

""",

response_description="Масив з історичними записами запитів користувача",

responses={

200: {"description": "Історію запитів успішно отримано"},

401: {"description": "Користувач не авторизований"},

500: {"description": "Помилка при читанні історії"}

}

)

async def get_task_history(

current_user: CurrentUserDep,

db_session: DBSessionDep

):

logs = await get_request_logs_user(db_session, current_user)

return [

{

"worker_info": log.worker_info,

"generated_info": log.generated_info,

"created_at": log.created_at

}

for log in logs

]

Отримує всі збережені запити та результати генерації, пов’язані з поточним авторизованим користувачем.

- current_user: авторизований користувач

Ендпоінт повертає список словників з worker_info, generated_info та created_at

Бізнес-логіку в проекті можна розбити на дві категорії: залежності та сервіси. Залежності – це функції, спрямовані на постачання або перевірку даних, які даються в аргументі до функції [30]. Цей функціонал зручно використовувати при валідації даних, які подаються до API. Проект містить дві категорії залежностей: залежності, які валідують дані користувача та ті, які перевіряють дані його авторизації (пароль, реєстарцію тощо).

```
import logging
from fastapi import HTTPException, Request

from app.schemas.auth import Signup, LoginArgs, LoginValidationResult
from app.models.user import User
from app.crud.user import get_user_by_username, get_user_by_email
from app.errors import Abort
from app.constants import ACCESS_TOKEN_TYPE
from app.utils.auth import verify_password, is_protected_username, utc_now
from app.services.auth import check_auth_user_from_token_by_payload, get_token_payload

from .user import CurrentUserDep
from .core import DBSessionDep


logging.basicConfig(level=logging.DEBUG)
```

```
async def validate_is_authenticated(current_user: CurrentUserDep) -> User:
    return current_user
```

Залежність FastAPI для ендпоінтів, щоб перевіряти авторизацію користуача перед запуском функції яка підв'язана під ендпоїнт де current_user залаженість з модуля для отриманння користувача

Залежність повертає модель БД користувача.

```
async def validate_signup(signup: Signup, db_session: DBSessionDep) -> Signup:
    logging.debug(f"Validating signup data: {signup}")
    if is_protected_username(signup.username):
        logging.debug(f"Invalid username detected: {signup.username}")
        raise HTTPException(status_code=400, detail="Invalid username")

    if await get_user_by_username(db_session, signup.username):
        logging.debug(f"Username already exists: {signup.username}")
        raise HTTPException(status_code=400, detail="Username already exists")

    if await get_user_by_email(db_session, signup.email):
        logging.debug(f"Email already  exists: {signup.email}")
        raise HTTPException(status_code=400, detail="Email already exists")

    return signup
```

Залежність FastAPI, яка перевіряє користувача реєстраційних даних:

– signup: реєстраційні дані, які потрібно підтвердити;
– db_session: залежність від сеансу бази даних.

Залежність повертає підтверджені реєстраційні дані.

Якщо ім'я користувача невірне, він вже існує, або якщо адреса вже існує, залежність викликає вийняток HTTPException.

```
async def validate_login(
        request: Request,
        login: LoginArgs,
        db_session: DBSessionDep
) -> LoginValidationResult:
    if not (user := await get_user_by_email(db_session, login.email)):
        raise HTTPException(status_code=401, detail="user-not-found")

    if not verify_password(login.password, user.hashed_password):
        raise Abort("auth", "invalid-password")

    refresh_token_id = None

    access_token = request.session.get("access_token")
    if access_token:
        payload = get_token_payload(token=access_token, check_expired_token=False)
        print(payload)
        refresh_token_id = payload.get("refresh_token_id")
        print(refresh_token_id)

    return LoginValidationResult(user=user, refresh_token_id=refresh_token_id)
```

Залежність FastAPI яка перевіряє користувача за даними логін форми.

request: request-об'єкт

login: pydantic-схема Login

db_session: сесія БД

```
def validate_password_reset(
        current_user: CurrentUserDep
) -> User:
if current_user.password_reset_expire:

        print(utc_now(), current_user.password_reset_expire)
        if utc_now() > current_user.password_reset_expire:
        raise Abort("auth", "reset-valid")

return current_user
```

Залежність FastAPI перед зміною паролю current_user: залаженість з модуля для отриманння користувача

Тепер оглянемо файл dependencies/user.py, який містить залежності, пов’язані з користувачами.

```
from typing import Annotated
from fastapi import Depends, HTTPException, status, Request


from app import models
from app.constants import ACCESS_TOKEN_TYPE, REFRESH_TOKEN_TYPE
from app.api.dependencies.core import DBSessionDep
from app.crud.user import get_user_by_email, get_admin_user_by_email
```

```
from app.schemas.auth import TokenData
from app.schemas.user import CurrentUserDataWithRefreshTokenID
from app.services.auth import get_access_token, get_email_from_token_payload, get_token_payload
from app.errors import credentials_exception


async def get_current_user(token: Annotated[str, Depends(get_access_token)], db_session: DBSessionDep) -> models.User:
    try:
        email = get_email_from_token_payload(token)
        token_data = TokenData(email=email)
    except Exception as e:
        raise credentials_exception

    user = await get_user_by_email(db_session, token_data.email)
    if user is None:
        raise credentials_exception

    return user
```

Залежність FastAPI для перевірки авторизації користувача проходячи різні етапи через сервіс auth

token: access токен із сеccії користувача

db_session: сеccія БД

Залежність повертає модель БД користувача.

```
async def get_current_user_with_refresh_token_id(token: Annotated[str, Depends(get_access_token)], db_session: DBSessionDep) -> CurrentUserDataWithRefreshTokenID:
    try:
        payload = get_token_payload(token, check_expired_token=False)
```

```
            refresh_token_id = payload.get("refresh_token_id")
            email = payload.get("sub")
            token_data = TokenData(email=email)
        except Exception as e:
            print(e)
            raise credentials_exception

        user = await get_user_by_email(db_session, email)
        if user is None:
            raise credentials_exception

        return CurrentUserDataWithRefreshTokenID(user=user,
refresh_token_id=refresh_token_id)
```

Залежність FastAPI для отримання рефреш токена - автентифікованого користувача

token: access токен із сеccії користувача

db_session: сеccія БД

Залежність повертає Pydantic-схему CurrentUserDataWithRefreshTokenID

```
    async def get_admin_user(token: Annotated[str,
Depends(get_access_token)], db_session: DBSessionDep) -> models.User:
        try:
            email = get_email_from_token_payload(token)
            token_data = TokenData(email=email)
        except Exception as e:
            raise credentials_exception

        user = await get_admin_user_by_email(db_session, token_data.email)
```

```
if user is None:
        raise credentials_exception

return user
```

Залежність FastAPI для перевірки авторизації ADMIN користувача проходячи різні етапи через сервіс auth

token: access токен із сессії користувача

db_session: сессія БД

Залежність повертає модель БД користувача.

Тепер оглянемо додаткові функції, які займаються аналізом даних та аутентифікацією (створення та управління токенами).

```
def new_token():
return secrets.token_urlsafe(32)
```

Генерує нових випадковий токен

```
def get_access_token(request: Request):
token = request.session.get('access_token')

if not bool(token):
        raise HTTPException(status_code=401, detail="Invalid access token")

return token
```

Отримує access-токен із сесії користувача.

request: HTTP-запит FastAPI з активною сесією.

Функція повертає токен доступу (JWT) як рядок.

Якщо токен відсутній або недійсний, функція викликає вийняток HTTPException.

```
def get_email_from_token_payload(token: str | bytes) -> str:
payload = decode_jwt(token)
if payload.get("type") != ACCESS_TOKEN_TYPE:
        raise credentials_exception

email = payload.get("sub")
if email is None:
        raise credentials_exception

return email
```

Отримує email користувача з JWT-токена (payload["sub"]).

token: jwt токен у вигляді рядка або байтів.

Функція повертає email користувача (sub).

Якщо токен має неправильний тип або не містить email, функція викликає вийняток HTTPException.

```
def decode_jwt(
        token: str | bytes,
        public_key: str = settings.auth_jwt.public_key_path.read_text(),
        algorithm: str = ACCESS_TOKEN_ALGORITHM,
        verify_exp: bool = True
) -> dict:
decoded = jwt.decode(
        token,
```

```
        public_key,
        algorithms=[algorithm],
        options={"verify_exp": verify_exp}
    )
    return decoded
```

Декодує JWT токен з перевіркою підпису та (опційно) терміну дії.

token: jwt-токен у вигляді рядка або байтів.
public_key: публічний ключ для перевірки підпису.
algorithm: алгоритм шифрування (за замовчуванням RS256 або HS256).
verify_exp: чи потрібно перевіряти термін дії токена.
Функція повертає розкодований словник payload.
Якщо токен невалідний, функція викликає вийняток jwt.InvalidTokenError.

```
    def encode_jwt(
        payload: dict,
        private_key: str = settings.auth_jwt.private_key_path.read_text(),
        algorithm: str = ACCESS_TOKEN_ALGORITHM,
        expire_minutes: int = settings.auth_jwt.access_token_expire_minutes,
        expire_timedelta: timedelta | None = None
    ) -> str:
    to_encode = payload.copy()
    now = datetime.utcnow()

    if expire_timedelta:
        expire = now + expire_timedelta
    else:
        expire = now + timedelta(minutes=expire_minutes)
```

```
    to_encode.update(
        exp=expire,
        iat=now,
        jti=str(uuid.uuid4()),
    )
    encoded = jwt.encode(
        to_encode,
        private_key,
        algorithm=algorithm
    )
    return encoded
```

Закодує JWT-токен з підписом, встановленням `exp`, `iat` і `jti`.

payload: основні дані токена.

private_key: приватний ключ для підпису.

algorithm: алгоритм шифрування (RS256, HS256).

expire_minutes: час життя токена у хвилинах (якщо не задано `expire_timedelta`).

expire_timedelta: час життя токена як об'єкт timedelta (перебиває `expire_minutes`).

Функція повертає підписаний JWT у вигляді рядка.

```
    def get_token_payload(
        token: str,
        check_expired_token: bool = True
    ) -> dict:
    try:
```

```
            payload = decode_jwt(token=token, verify_exp=check_expired_token)
        except jwt.InvalidTokenError as e:
            raise HTTPException(
            status_code=401,
            detail=f"Invalid token error: {e}"
            )
        return payload
```

Декодує JWT та отримати payload із токена з перевіркою терміну дії.

token: jwt access/refresh токен.

check_expired_token: чи перевіряти термін дії (exp).

Функція повертає payload у вигляді словника.

Якщо токен невалідний або підпис неправильний, функція викликає вийняток HTTPException.

```
        def validate_token_type(
            payload: dict,
            token_type: str,
        ) -> bool:
        current_token_type = payload.get(TOKEN_TYPE_FIELD)
        if current_token_type == token_type:
            return True
        raise HTTPException(
            status_code=401,
            detail=f"Invalid token type {current_token_type!r} expected {token_type!r}"
        )
```

Перевіряє тип токена у payload – чи відповідає очікуваному.

payload: розкодований payload токена.

token_type: очікуваний тип токена (наприклад, "access", "refresh").

Функція повертає True, якщо тип відповідає.

Якщо тип токена не співпадає з очікуваним, функція викликає вийняток HTTPException.

```
def check_auth_user_from_token_by_payload(
token_type: str,
user_email: str,
payload: dict,
) -> bool:
try:
        validate_token_type(payload, token_type)
        if payload.get("sub") != user_email:
        return False
        return True
except Exception as e:
        return False
```

Перевіряє, чи співпадає email у payload з поточним користувачем і чи тип токена валідний.

token_type: очікуваний тип токена ("access" або "refresh").

user_email: email користувача, що виконує запит.

payload: payload із JWT.

Функція повертає True, якщо токен валідний і email співпадає. False — інакше.

```
def create_jwt(
        token_type: str,
        token_data: dict,
        expire_minutes: int = settings.auth_jwt.access_token_expire_minutes,
        expire_timedelta: timedelta | None = None,
) -> str:
jwt_payload = {TOKEN_TYPE_FIELD: token_type}
jwt_payload.update(token_data)
return encode_jwt(
        payload=jwt_payload,
        expire_minutes=expire_minutes,
        expire_timedelta=expire_timedelta,
)
```

Створює JWT-токен заданого типу з переданим payload.

token_type: тип токена — "access" або "refresh".
token_data: дані, які необхідно вбудувати у payload.
expire_minutes: термін дії у хвилинах (опціонально).
expire_timedelta: альтернативний спосіб задання терміну дії.
Функція повертає створений підписаний JWT токен.

```
def create_access_token(user: User, refresh_token: AuthToken) -> str:
jwt_payload = {
        "sub": user.email,
        "refresh_token_id": str(refresh_token.id)
}
return create_jwt(
```

```
        token_type=ACCESS_TOKEN_TYPE,
        token_data=jwt_payload,
        expire_minutes=settings.auth_jwt.access_token_expire_minutes,
    )
```

Створює JWT access-токен для вхідного користувача з посиланням на refresh-токен.

user: користувач, для якого створюється токен.

refresh_token: об'єкт refresh-токена.

Функція повертає access-токен (JWT) у вигляді рядка.

```
def create_refresh_token(user: User) -> str:
jwt_payload = {
        "sub": user.email,
}
return create_jwt(
        token_type=REFRESH_TOKEN_TYPE,
        token_data=jwt_payload,

expire_timedelta=timedelta(days=settings.auth_jwt.refresh_token_expire_da
ys),
)
```

Створює JWT refresh-токен для користувача.

user: користувач, email якого буде використано в payload.

Функція повертає refresh-токен у вигляді підписаного JWT.

```
async def get_worker_statistics(file: UploadFile):
content = await file.read()
return await __get_worker_statistics(content)

async def get_task_by_cv(file: UploadFile):
content = await file.read()
text = __byte_to_text(content)

async def get_worker_data(text: str):
return await __get_worker_data(text)
return await __get_worker_data(text, return_text=True)
```

Функції-”враппери”, які служать як “міст” між API та функціоналом.

file: uploadFile(вбудований тип FastAPI, зміст якого читається за допомогою file.read()).

```
async def __get_worker_statistics(content: bytes):
client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))
reader = PdfReader(io.BytesIO(content))
text = ""
for page in reader.pages:
        text += page.extract_text()

response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
        {
                "role": "system",
                "content": (
```

```
                "Це інформація з документу українською мовою. Проаналізуй його й поверни рекомендацію "
                "щодо того, яку працю можна запропонувати працівнику у форматі JSON "
                "({company_name, name, surname, qualifications, years_of_experience, justification}). Всі поля повинні існувати, якщо одно з них не можна заповнити, запиши туди null. Якщо жодного з перечня можливих становищ не можна вибрати, напиши."
                "Перечень можливих становищ: " + get_jobs() +
                " justifiction зроби як мінімум на 750 слів. Поверни тільки JSON. Повертай виключно українською."
                ),
            },
            {"role": "user", "content": text},
            ],
            stream=False
        )
        stuff = re.sub(r"^```(?:json)?\s*|\s*```$", "", response.choices[0].message.content, flags=re.MULTILINE).strip()
        return json.loads(stuff), text
```

Прочитати PDF-файл працівника, витягнути текст і передати його в OpenAI для аналізу.

content: вміст PDF-файлу у вигляді байтів.

Функція повертає Рекомендацію OpenAI у форматі словника JSON з наступними полями:

- id: id запропонованої роботи зі списку;
- name: ім'я працівника;
- surname: прізвище;
- qualifications: основні навички;

– justification: детальний коментар щодо рекомендації.

```
async def __get_worker_data(text: str, return_text: bool = False):
client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
        {
                "role": "system",
                "content": (
                "Це інформація про працівника. Проаналізуй його й поверни завдання, яке йому найбільш пасує, "
                "в форматі JSON. Завдання, які потребує підприємство: " + get_tasks() +
                " Впевнись, що його навички відповідають завданням. Якщо працівнику не можна приділити жодного завдання, поверни пустий JSON. Поверни тільки JSON."
                ),
        },
        {"role": "user", "content": text},
        ],
        stream=False
)
stuff = re.sub(r"^```(?:json)?\s*|\s*```$", "", response.choices[0].message.content, flags=re.MULTILINE).strip()

if return_text:
        return json.loads(stuff), text
```

return json.loads(stuff)

Надіслати текстову інформацію про працівника до OpenAI для генерації персоналізованого завдання.

text: повна текстова інформація про працівника (включаючи навички, освіту тощо).

Функція повертає завдання, яке найкраще підходить працівнику, у форматі JSON:

– task_id: id завдання;
– task_title: назва завдання;
– reason: чому саме це завдання пасує працівнику.

```
def __byte_to_text(content: bytes) -> str:
reader = PdfReader(io.BytesIO(content))
text = ""
for page in reader.pages:
        text += page.extract_text()

return text
```

Видає текст PDF файлу виходячи с представлених байтів

content: передані bytes документа

Функція повертає змінну типу str: текст с PDF документа.

3.3 Розроблення інтерфейсу

api/docs – Вхідна точка програми (рис. 3.2).

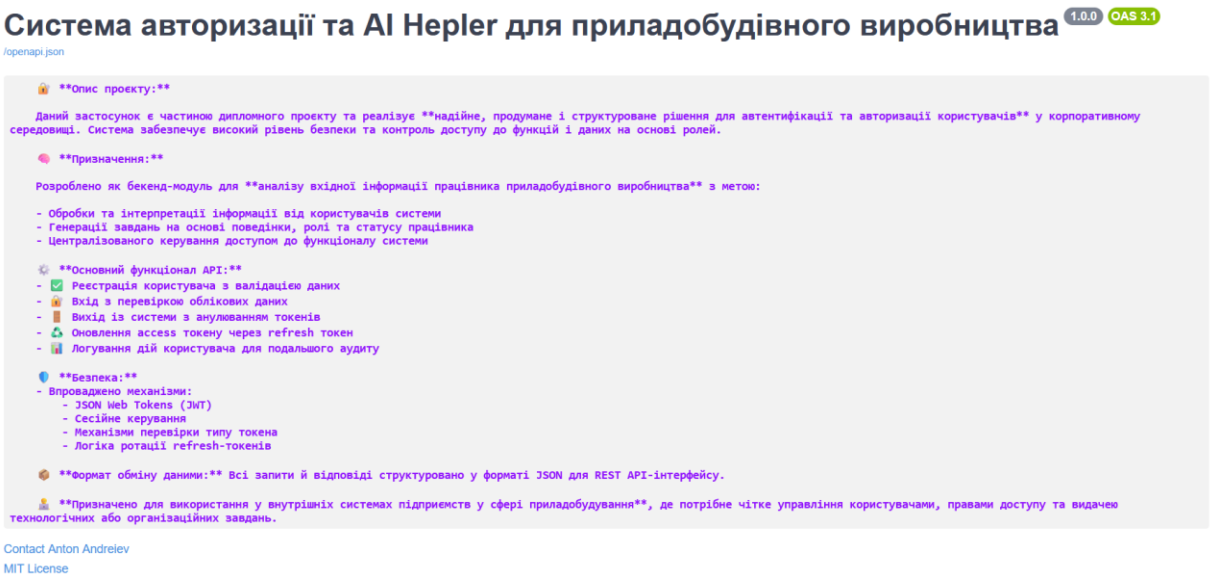

Рисунок 3.2 – Опис програми

api/docs#/Auth%20System/signup_auth_registration_post – реєстрація користувача (рис. 3.3).

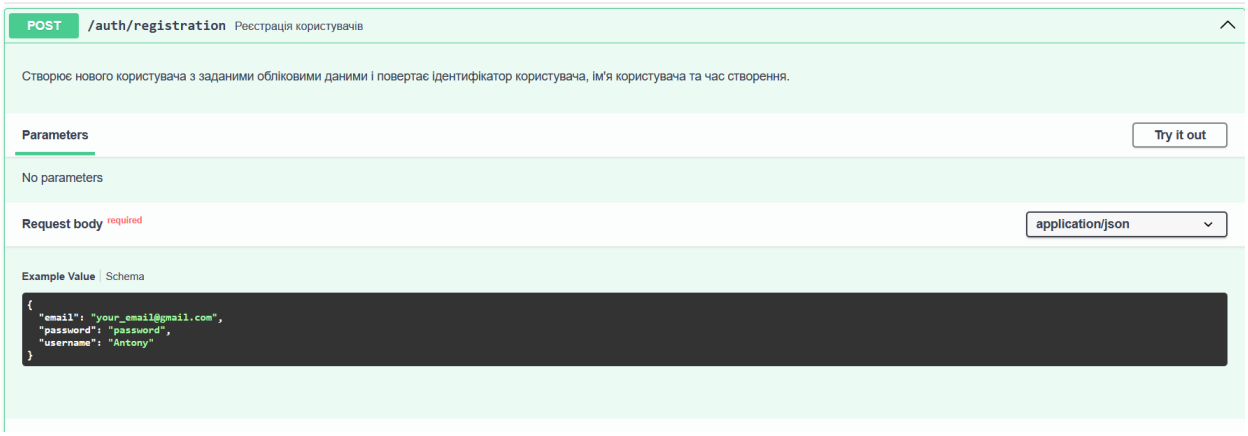

Рисунок 3.3 – Реєстрація

Можемо виконати запрос реєстрації за допомогою кнопки “Try it out”. Результатом виконання запиту є відповідь про успішну реєстрацію користувача (рис. 3.4).

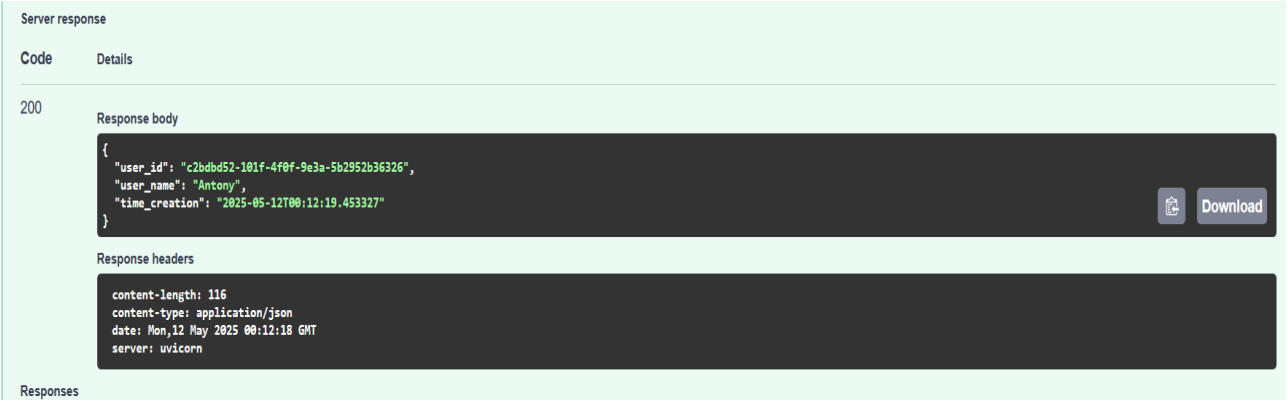

Рисунок 3.4 – Відповідь системи про успішну реєстрацію користувача

api/docs#/Users Control System/user_details_api_users_me_get – за допомогою цього ендпоінту користувач може перевірити, чи він залогований (рис. 3.5).

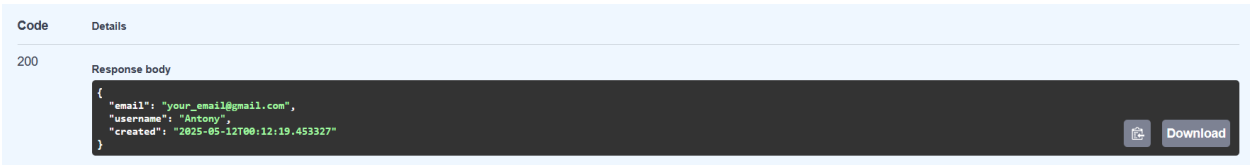

Рисунок 3.5 – Інформація про користувача

Функціонал аутентифікації та управління профілем розділений на різні підрозділи, що спрощує роботу з застосунком (рис. 3.6).

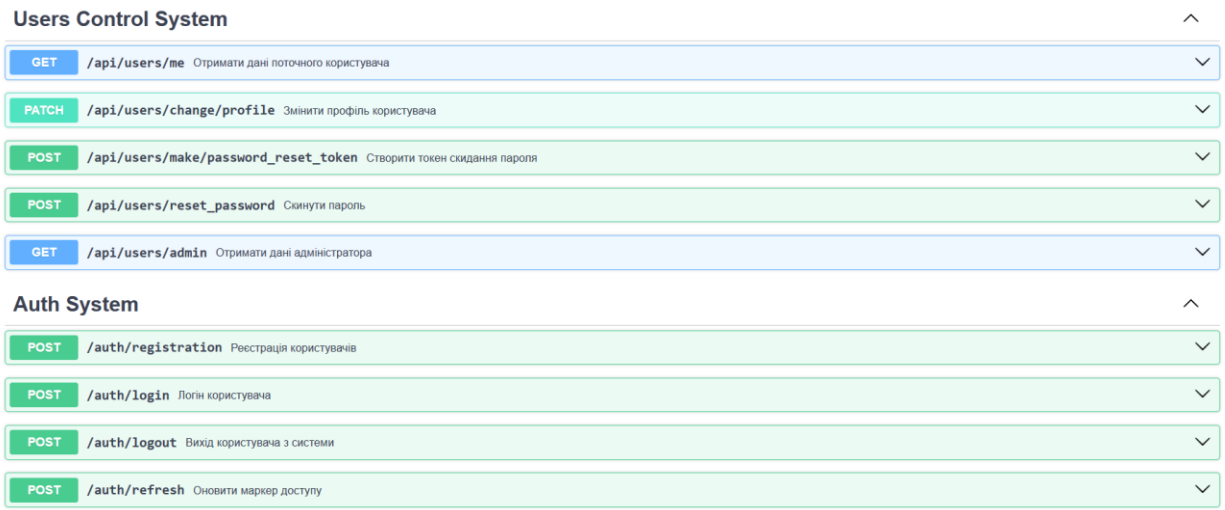

Рисунок 3.6 – Підрозділи управління профілем та авторизація

Спробуємо проаналізувати резюме. Для цього треба просто загрузити файл з комп’ютера користувача до веб-додатку. На виході отримуємо інформацію щодо становища, яке найбільш підходить працівнику (рис. 3.7).

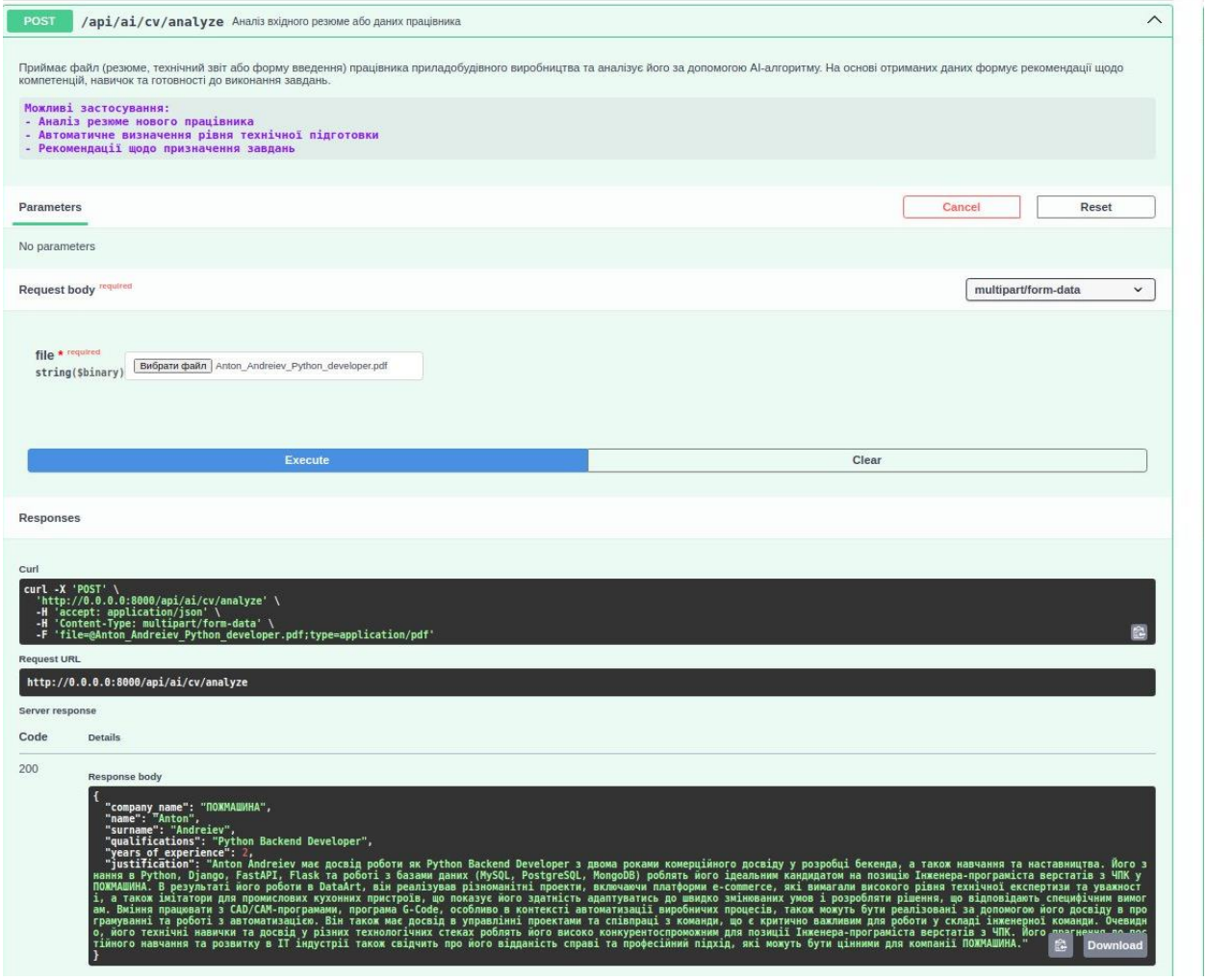

Рисунок 3.7 – Приклад роботи ендпоінта, який аналізує дані

Також можемо використати додаток, щоб перевірити історію аналізу. Це може бути корисним, для подальшого аналізу (Наприклад, кількості охочих працювати на виробництві) (рис. 3.8).

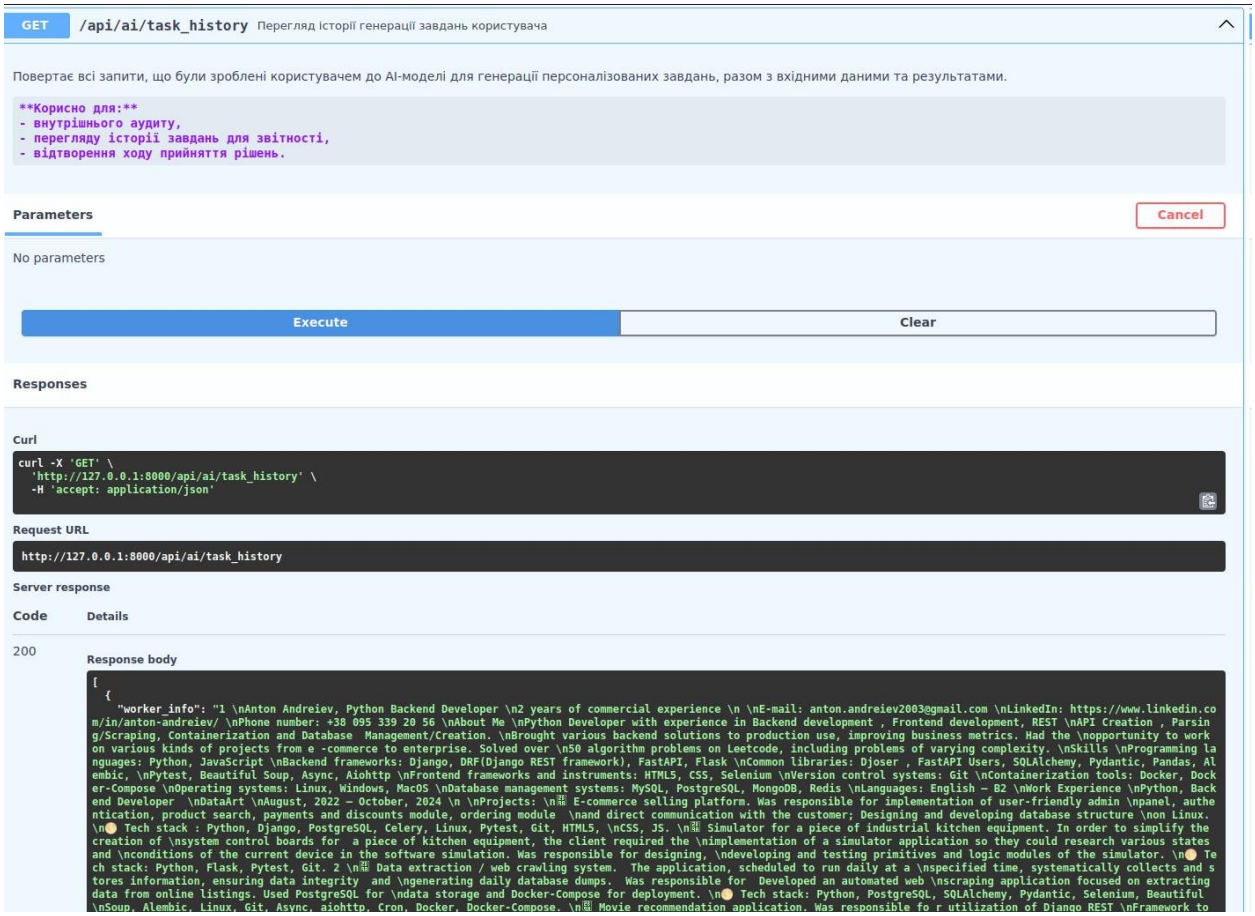

Рисунок 3.8 – Історія видавання задач

Як можна побачити, такий інтерфейс є відносно простим та інтуіційним для використання, приймаючи, що користувач має базовий рівень технічних знань.

3.4 Охорона праці

Робота зі створення та впровадження сервісу FastAPI-LLM для аналізу вхідної інформації працівника приладобудівного виробництва ведеться в умовах, де основні небезпечні й шкідливі фактори пов’язані з використанням комп’ютерної техніки, електропостачанням і серверним обладнанням, інформаційною безпекою й психоемоційним навантаженням операторів. Відповідно до Закону України «Про охорону праці», ДСанПіН 3.3.2-017-2011 і ДБН В.1.1-7:2016 робоче місце розробника та адміністратора сервісу слід облаштувати ергономічним робочим столом і кріслом із регулюваннями;

екран монітора розташовується на відстані 500–700 мм, верхній край – на рівні очей, яскравість та контрастність відрегульовані для виключення стробоскопічного ефекту. Загальне освітлення – не менше 300 лк, локальне – 500 лк; мікроклімат у межах 22 ± 2 °C, відносна вологість 40–60 %. Для зниження психофізіологічних навантажень вводяться регламентовані перерви – 10 хв щогодини при роботі з кодом і LLM-чатом, що відповідає гігієнічним рекомендаціям.

Електробезпека забезпечується живленням обладнання від мережі 220 В із системою заземлення TN-S; усі корпуси серверів і металеві частини стояків під'єднані до захисного провідника, а диференційні вимикачі (30 мА) та автоматичні вимикачі захищають від коротких замикань та струмів витоку. Джерело безперебійного живлення класу VFI-SS-111 підтримує роботу сервісу та коректне завершення сесій LLM при аварійному відключенні. Кабель-менеджмент організовано у фальш підлозі з негорючих матеріалів, категорія пожежної небезпеки приміщення – В-4; встановлено вуглекислотний вогнегасник ВВК-2 (за ПУЕ та ДСТУ EN 3-7). Система вентиляції підтримує тепловиділення серверів < 400 Вт/м², шум < 50 дБ (А) [31].

Захист даних і профілактика кібер ризиків входять до програмних заходів охорони праці, оскільки витоки персональних даних і хибні AI-рекомендації можуть спричинити виробничі травми. Ієрархія контролю включає: багатофакторну аутентифікацію (OAuth2 + JWT), сегментацію мережі VLAN, SSL-шифрування, журналювання API-викликів, резервне копіювання в ізольоване сховище та регулярний пентест згідно з ISO 27001. LLM-моделі проходять процедури RAG-фільтрації та human-in-the-loop-перегляду, що знижує ризик видачі небезпечних або дискримінаційних інструкцій працівникам. Swagger-інтерфейс доступний лише через внутрішній VPN і містить дозовану документацію для запобігання інформаційному перевантаженню операторів.

У виробничому середовищі система мінімізує фізичні ризики, автоматично призначаючи завдання з урахуванням актуального стану

працівника й обладнання, чим скорочує непотрібні переміщення персоналу, усуває дублювання заявок і зменшує ймовірність помилок, спричинених людським фактором. Разом із тим, кожному користувачеві надаються інструкції з безпечної роботи з терміналом, а вбудований модуль нагадувань відповідає за своєчасне проходження обов’язкових інструктажів. Аналіз ризиків методом PHA показав, що після впровадження зазначених інженерних і організаційних заходів імовірність та тяжкість залишкових ризиків класифікуються як прийнятні (ALARP), що підтверджує відповідність проекту вимогам нормативної бази щодо охорони праці та безпечної експлуатації інформаційних систем.

ВИСНОВКИ

В результаті написання першого розділу кваліфікаційної проведено аналіз використання автоматичних технологій та штучного інтелекту в виробництві, першочергово в аналізі даних. Проаналізовані потреби та розв’язання, пов’язані з аналізом даних за допомогою штучного інтелекту, а також описаний процес аналіза даних з допомогою API.

У ході написання другого розділу атестаційної роботи було проведено аналіз доступних розв’язань створення веб-застосунків та обрано відповідні технології до проекту. Зокрема, була обрана мова програмування Python та фреймворк FastAPI, за допомогою яких було збудовано веб-додаток, який служить інтерфейсом для аналізу даних користувача.

Було проведено аналіз ринку LLM та обрано модель, яка дозволяє проводити аналіз вхідної інформації достатньо щільно. Також був імплементований функціонал передачі даних до LLM та підібраний доцільний запит(prompt).

В третьому розділі атестаційної роботи було створено веб-застосунок, який дозволяє аналізувати дані кандидатів на працю на приладобудівному виробництві та приділяти їм відповідні завдання.

Також було створено систему авторизації користувача, що дозволяє значно знизити навантаження на систему, даючи доступ до неї тільки обраним користувачам.

Врешті, було створено зрозумілий та простий в користуванні інтерфейс, який дозволяє особам з технічними знаннями використання функціоналу додатку.

Були виконані наступні завдання:

- проаналізовано стан автоматизації на виробництвах;
- розглянуто історію розвитку штучного інтелекту;
- описано концепт LLM та їх структуру;

- виділено плюси й мінуси використання LLM на виробництві;
- обрано платформи й технології, які будуть застосовані.
- обґрунтовано вибір платформ та технологій;
- визначено структуру програмного забезпечення для аналізу інформації;
- розроблено інтерфейс користувача програми;
- визначено оптимальний спосіб аналізувати вхідні дані;
- розроблено програму для аналізу вхідної інформації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методичні вказівки з підготовки кваліфікаційної роботи бакалавра для здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології освітньої програми «Системна інженерія» / Упоряд.: І.Ш. Невлюдов, О.М. Цимбал, О.В. Токарєва, А.І. Бронніков. – Харків: ХНУРЕ, 2022. – 66 с.

2. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки. структура та правила оформлення. Введ. 2015-06-22. К. Держстандарт України, 2017. – 29 с.

3. Методичні вказівки з підготовки кваліфікаційної роботи бакалавра для студентів усіх форм навчання спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» освітньої програми «Автоматизація та комп'ютерно-інтегровані технології» / Упоряд.: І.Ш. Невлюдов, А.О. Андрусевич, О.В. Токарєва, С.П. Новоселов, О.В Сичова. Харків: ХНУРЕ, 2022. – 55 с.

4. Андреєв А. С. Розроблення програмного забезпечення для аналізу вхідної інформації робітника приладобудівного виробництва для видачі завдань на виконання / А. С. Андреєв // Автоматизація та Приладобудування («Automation and Development of Electronic Devices» ADED-2025) : збірник студентських наукових статей. – Харків : ХНУРЕ, 2025. – Вип. 1. – С. 7-11.

5. Андреєв А.С. Штучний інтелект та машинне навчання в автоматизації / А. С. Андреєв // Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки 2024 : тези доповідей І-ої Всеукр. конф., 16-17 травня 2024. – Харків, 2024. – С. 45-49.

6. Андрєєв А. С. Особливості створення семантичних мереж / А. С. Андрєєв // Автоматизація та приладобудування = Automation and Development of Electronic Devices (ADED'2021) [Електронний ресурс]: зб. студ. наук. ст. – Харків : ХНУРЕ, 2021. – Вип. 2. – С. 89–92.

7. Андреєв А.С. Аналіз особливостей розробки WEB-додатків / А. С. Андреєв // Automation and Development of Electronic Devices (ADED'2022) : collection of Students' Scientific Paper. – Kharkiv : Kind of Kharkiv National Universitety of Radio Elektronics [electronic edition], 2022. – Part 2. – P. 45-49.

8. Андреєв А.С. Перспективи використання PHP та MYSQL в проектах / А. С. Андреєв // Automation and Development of Electronic Devices (ADED'2023) : collection of Students' Scientific Paper. – [Electronic edition]. – Kharkiv: Kind of Kharkiv National Universitety of Radio Elektronics, 2023. – Part 1. – P. 66-69.

9. Sotnik S. QR codes in production / S. Sotnik, A. Andreiev // Manufacturing & Mechatronic Systems 2023 : proceedings of the VIIst International Conference, Kharkiv, October 19-20, 2023. – Kharkiv, 2023. – P. 19-21.

10. Sotnik S. V. Gamification in science: game platforms for learning / S. V. Sotnik, A. S Andreiev // Комп'ютерні ігри і мультимедіа як інноваційний підхід до комунікації : матеріали III Всеукр. наук.-техн. конф. молодих вчених, аспірантів і студентів, 28-29 вересня 2023 р. – Одеса, 2023. - С. 87-89.

11. Андреєв А. С. Пошук інформації в інтернеті: проблеми та можливості / А.С. Андреєв // Автоматизація та Приладобудування = Automation and Development of Electronic Devices (ADED'2023) : збірник студентських наукових статей. – Харків : ХНУРЕ, 2024. – Вип. 1. – С. 116-121.

12. Andreiev A. S. Analysis of robotics platforms for educational and research purposes / A. S. Andreiev, S. V. Sotnik // Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації - 2024 : матеріали IV Всеукр. наук.-техн. конф. молодих вчених, аспірантів і студентів, Одеса, 26-27 вересня, 2024. - Одеса : ОНТУ, 2024. – P. 25-27.

13. Andreiev A. S. Computer games and Web design / A. S. Andreiev, S. V. Sotnik // Інформаційні технології і автоматизація – 2024 : матеріали XVII міжнародної науково-практичної конференції, 31 жовтня-1 листопада 2024 р. – Одеса : Видавництво ОНТУ, 2024 р. – С. 712-714.

14. Andreiev A. Comparative analysis of robotics platform: Webots, Coppeliasim and Gazebo / A. Andreiev, S. Sotnik // Сучасні проблеми і

досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій : тези доповідей XII Міжнародної науково-практичної конференції, 10-12 грудня 2024 р. - Запоріжжя: НУ «Запорізька політехніка», 2024. – С. 96-100.

15. Харківський національний університет радіоелектроніки [Електронний ресурс] /– Режим доступу: www / URL: https://nure.ua/faculty/fakultet-avtomatiki-i-komp-yuterizovanih-tehnologiy

16. National library of medicine : Artificial Intelligence-Assisted Surgery: Potential and Challenges [Електронний ресурс] /– Режим доступу: www / URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC7768095/ (дата звернення 27.05.2025)

17. Світова статистика використання ШІ бізнесами [Електронний ресурс] /– Режим доступу: www / URL: https://www.marketing-ua.com/article/shi-instrumenti-dlya-rozvitku-ecommerce-biznesu/ (дата звернення 27.05.2025)

18. Документація Django [Електронний ресурс] /– Режим доступу: www / URL: https://docs.djangoproject.com/en/5.2/intro/overview/ (дата звернення 27.05.2025)

19. Документація Flask [Електронний ресурс] /– Режим доступу: www / URL: https://flask.palletsprojects.com/en/stable/quickstart/#a-minimal-application (дата звернення 27.05.2025)

20. Документація FastAPI [Електронний ресурс] /– Режим доступу: www / URL: https://fastapi.tiangolo.com/ (дата звернення 27.05.2025)

21. Документація Pydantic [Електронний ресурс] /– Режим доступу: www / URL: https://docs.pydantic.dev/latest/ (дата звернення 27.05.2025)

22. Документація Pydantic [Електронний ресурс] /– Режим доступу: www / URL: https://docs.pydantic.dev/latest/why/#performance (дата звернення 27.05.2025)

23. Документація Uvicorn [Електронний ресурс] /– Режим доступу: www / URL: https://www.uvicorn.org/ (дата звернення 27.05.2025)

24. Документація sqlalchemy [Електронний ресурс] /– Режим доступу: www / URL: https://www.sqlalchemy.org/philosophy.html (дата звернення 27.05.2025)

25. The Python Package Index [Електронний ресурс] /– Режим доступу: www / URL: https://pypi.org/project/asyncpg/ (дата звернення 27.05.2025)

26. OpenAI [Електронний ресурс] /– Режим доступу: www / URL: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/ (дата звернення 27.05.2025)

27. Github /– Режим доступу: www / URL: https://github.com/openai/openai-python (дата звернення 27.05.2025)

28. Документація FastAPI [Електронний ресурс] /– Режим доступу: www / URL: https://fastapi.tiangolo.com/#interactive-api-docs (дата звернення 27.05.2025)

29. Statista [Електронний ресурс] /– Режим доступу: www / URL: https://www.statista.com/chart/16567/popular-programming-languages/ (дата звернення 27.05.2025)

30. Документація FastAPI [Електронний ресурс] /– Режим доступу: www / URL: https://fastapi.tiangolo.com/tutorial/dependencies/ (дата звернення 27.05.2025)

31. Закон України «Про охорону праці»: № 2694-XII від 14.10.1992 р. (зі змінами і доповненнями станом на 01.12.2024 р.) // Відомості Верховної Ради України. – 1993. – № 49.